LE MAGAZINE PRATIQUE DE LA PROGRAMMATION

8

PC FUTÉ PC FUTÉ PC FUTÉ PC FUTÉ PC FUTÉ PC FUTÉ

PC FUTÉ

Participez et gagnez au concours des créateurs

Initiation, graphisme, techniques de programmation, courrier des lecteurs trucs et astuces...

Visual Basic, Delphi, Langage C, Qbasic, DOS, Windows...

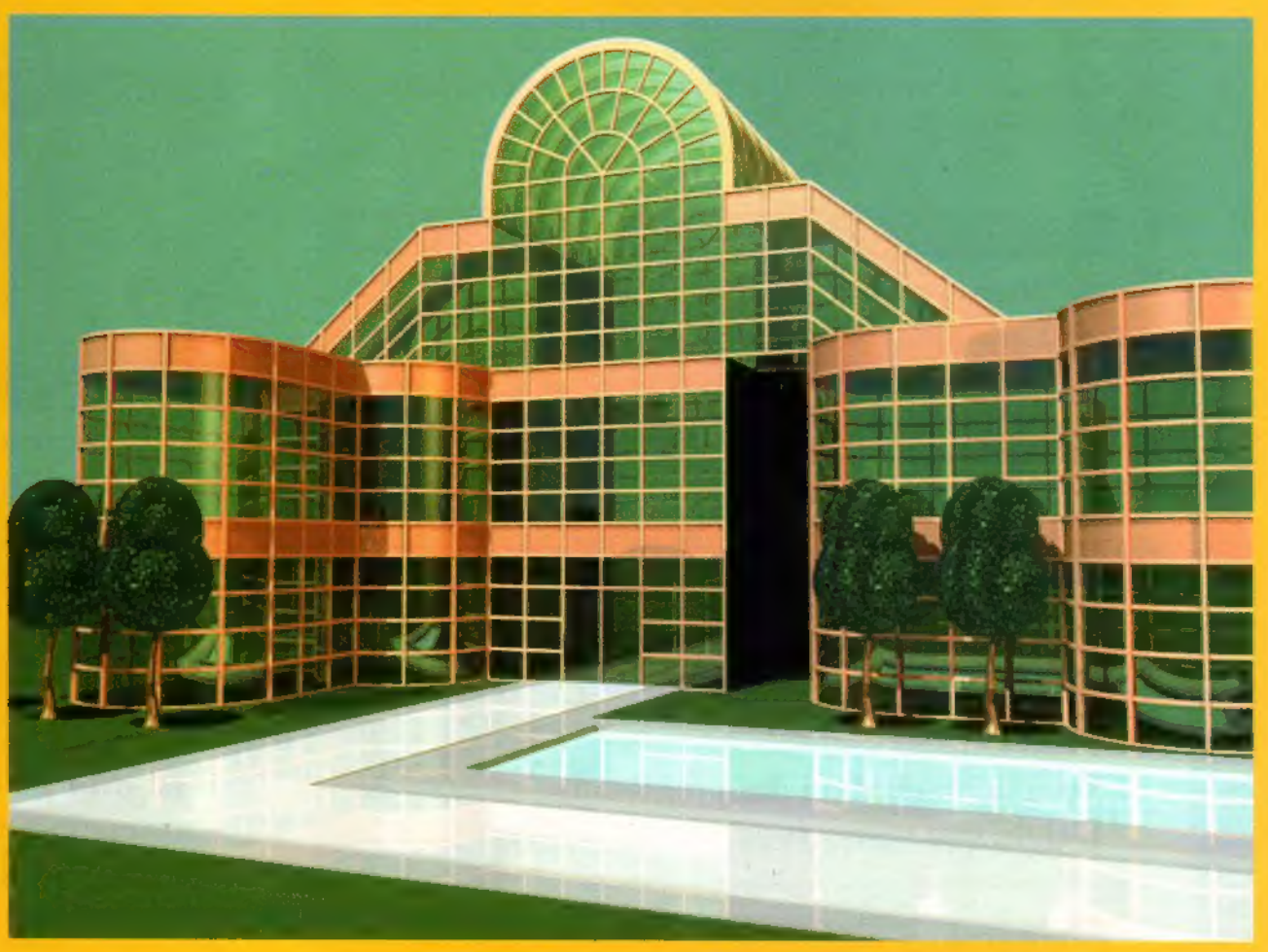
Delphi contre Visual Basic

Le combat du siècle?

Delphi
L'épreuve du feu
Création d'un logiciel en Delphi

Visual Basic
Initiation
Trucs et astuces

Langage C
Créez votre jeu de stratégie



Gratuit

la disquette du numéro

Les routines, programmes et listings du magazine
Des listings et des VBX pour Visual Basic et Delphi
Les créations de nos lecteurs.

Une disquette accompagne cet exemplaire sinon, demandez-la à la caisse.

PC FUTÉ N° 8 - Octobre/Novembre 1996

S O M M A I R E

ACTUALITES 4

• Les news
Le monde de l'informatique ne cesse de bouger ! De nouveaux produits sortent tous les jours !

COURRIER 5

• Vos questions, nos réponses
Quels que soient vos problèmes, n'hésitez pas à nous écrire. Ces pages sont ouvertes à tous.

CONCOURS 9

Avez-vous des idées originales sur la manière de créer un jeu ? Si oui, participez à notre concours !

ANNUAIRE 10

• Inscrivez-vous dans notre annuaire
Vous cherchez des contacts pour développer un projet intéressant ? Une seule solution, inscrivez-vous à notre annuaire informatique !

DEBUTANT 11

• Qu'est-ce qu'un langage de programmation ?
La création d'un programme ressemble plus à une recette de cuisine qu'à une conversation classique.

TECHNIQUES DE PROGRAMMATION 15

• Les jeux d'aventure
Les jeux d'aventure font partie des logiciels les plus complexes à programmer.

INITIATION QBASIC 19

• Trucs et astuces
Voici quelques trucs et astuces pour vous simplifier la vie.

INITIATION DELPHI 21

• L'épreuve du feu page 21
Delphi est-il un véritable concurrent pour le Visual Basic ? Pour connaître la réponse à cette question, nous avons mis un exemplaire de Delphi entre les mains d'un spécialiste en Visual Basic.

• Les fichiers Delphi page 23
Voici la description des programmes composant Delphi que nous avons mis sur la disquette.

JEU DELPHI 24

• Créez votre propre Columns
La puissance de Delphi rend aisée la réalisation de petits programmes graphiques.

INITIATION VISUAL BASIC 28

• Manipulations graphiques en VB 3.0 page 28
Le VB est assez limité du point de vue manipulations d'images. Heureusement, il est possible d'utiliser les fonctions graphiques de Windows !

• Trucs et astuces page 31
Quelques techniques pour vous faciliter la vie !

• Les logiciels VB page 34
Description des listings et des fichiers VBX contenus sur la disquette du mois.

JEU VISUAL BASIC 36

• Créez votre Pacman (III)
Le moment tant attendu est venu ! Nous allons donner l'étincelle de vie à la créature ! Le donjon va enfin avoir un gardien !

GRAPHISME ET LANGAGE C 39

• Accès direct à la mémoire vidéo page 39
La manière la plus efficace pour réaliser des logiciels graphiques performants est d'agir directement sur la mémoire vidéo.

• Chargement d'une image BMP page 41
Cet article vous explique comment charger une image BMP sur l'écran. C'est indispensable pour créer des logiciels graphiques.

JEU ET LANGAGE C 44

• Programmation d'un jeu de stratégie
Vous aimez Civilisation ? Vous avez des idées sur le jeu de simulation idéal ? Dans ce cas, pourquoi ne pas créer votre propre logiciel ?

LOGICIEL 50

• Utilisation de DirectDraw
Cet article vous présente un ensemble de routines permettant d'utiliser les fonctions du module graphique DirectX pour Windows 95.

LECTURES 57

• Les livres du mois
Il est difficile de choisir un ouvrage de programmation, tant le choix est vaste ! Voici notre sélection du mois.

INSTALLATION DISQUETTE 63

Comment installer la disquette et que contient-elle ?

© J.D. Press 1996 - Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit, des pages publiées dans la présente publication, faite sans l'autorisation de l'éditeur est illicite et constitue une contrefaçon. (Loi du 11 mars 1957 - art. 40 et 41 du Code Pénal art. 425).
PC Futé est une marque déposée par J.D. Press n° 95554304/INPI.
La rédaction n'est pas responsable des textes, illustrations, photos et dessins publiés qui engagent la seule responsabilité de leurs auteurs.

EDITO

L'évolution des langages de programmation se traduit par une complexité qui va croissante. C'est tout particulièrement le cas des langages *Windows* qui bénéficient de la puissance de l'interface graphique et surtout de sa structure interne complexe. Heureusement, nous assistons actuellement à l'émergence d'une nouvelle génération de langages comme *Visual Basic* et *Delphi*. Ces systèmes de développement sont faciles à utiliser et offrent la possibilité de créer rapidement des programmes *Windows*. Le principe de base est l'utilisation d'objets graphiques intelligents, capables d'effectuer certaines tâches par eux-mêmes.

Cela transforme la programmation en une sorte de jeu de mécano où le développeur choisit les objets qui lui plaisent, les place sur l'écran et écrit quelques instructions de manière à lier les objets entre eux et gérer les variables du programme. Cette technique révolutionnaire permet d'écrire des applications *Windows*, sans se soucier des détails techniques complexes. Ceux-ci sont traités automatiquement par les objets graphiques. Seul le créateur des objets doit se plonger dans les subtilités de la programmation *Windows*. Les langages, tels que *Delphi* et *Visual Basic*, sont précurseurs des langages du siècle prochain. Il est probable que les années à venir soient marquées par l'apparition d'autres langages utilisant les mêmes principes. Par exemple, pourquoi pas un *Delphi C*, permettant d'écrire des programmes C avec la même simplicité que *Delphi* ? On peut y rêver en regardant le *Visual C*...

P. L.

Gérant et directeur
de la publication :
Jérôme R. Virazels

Editeur/Responsable de la rédaction :
Serge Fenez

Rédacteur en chef :
Patrick Leclercq

Ont participé à la rédaction
de ce numéro :
Michel Legrand
Jacques Lemercier
Paul Leroy
Roch Lombardon
John Lynch
Gunther Reiner
Mike Terranova

Maquette et mise en page :
D. d'Inguimbert
92 300 Levallois - 47 48 95 51

Impression :
Litografia Roses S.A
08 850 Gavá - Barcelona

PC FUTÉ est édité par J.D. Press
24 bis, rue du Gabon
75012 Paris
tél. 44 73 00 11
fax 44 73 94 00
SARL de presse au capital de 10 000 F
R.C.S. Paris B 395 105 505

Principal Associé :
Jérôme R. Virazels

Commision Paritaire n° 76662
ISSN en cours
Dépôt légal à la parution
Périodicité : bimestrielle
Abonnement : F 468 (12 numéros)
Etranger : nous consulter

Prix de vente au numéro : F 49

Régie de publicité :
tél. 44 73 00 11
fax 44 73 94 00

Les news

LE MONDE DE L'INFORMATIQUE NE CESSE DE BOUGER ! DE NOUVEAUX PRODUITS SORTENT TOUS LES JOURS, TANDIS QUE LES ÉDITEURS MULTIPLIENT LES ANNONCES. VOICI UN RÉSUMÉ ACTUALITÉS QUI NOUS ONT PARU IMPORTANTES.

Shareware

L'aFAs (Association Française des Auteurs de Shareware) existe depuis novembre 1995. Ses objectifs sont d'aider les auteurs à améliorer la qualité de leurs logiciels, de mettre en contact les auteurs entre-eux et de promouvoir le shareware en France. Voici leur adresse, pour adhérer à l'association ou obtenir des renseignements :
Anne Marie Beignatborde
L'Association Française des Auteurs de Shareware
La Herrane
32800 Réans
Tél: 62 09 74 85
Internet: afas@sunsnet.dedal.fr.net

Matériels

Le Pentium 75 n'est plus fabriqué. Le nouveau microprocesseur de base est le Pentium 100. De plus, Intel a annoncé, qu'à la fin de l'année, le microprocesseur de base serait le Pentium 133. Encore quelques mois et les possesseurs de Pentiums 75 pourront mettre des Pentiums 133 dans leur machine pour le prix d'un baril de lessive !
Le prix de la mémoire s'est effondré en quelques mois. On trouve actuellement des barrettes 8 Mo EDO pour moins de 400 F et des barrettes 16 Mo pour moins de 800 F. C'est le moment d'augmenter la mémoire pour profiter pleinement de Windows 95 qui n'est vraiment utilisable qu'avec 16 Mo !
La carte graphique Matrox Mystic est en vente depuis peu. C'est la première carte conçue spécifiquement pour exploiter le système graphique DirectX pour Windows 95. Son prix théorique est de 1 490 F en version 2 Mo, mais certaines boutiques parisiennes la proposent déjà pour moins de 1 000 F.
Les cartes graphiques dialoguent avec les PC par l'intermédiaire du bus PCI qui fonctionne à 66 Mhz. Cela va changer, avec la sortie prochaine d'un nouveau bus graphique dont la vitesse de transfert est 4 fois supérieure à celle du bus PCI. Nous en reparlerons prochainement.

Visual Basic

Le Visual Basic 5.0 devrait voir le jour d'ici quelques mois. Par rapport à la version 4.0, il devrait enfin être capable de produire des programmes exécutables, n'ayant plus besoin de runtime. Nous en reparlerons ultérieurement.
La société Micrografx, connue pour sa gamme de logiciels graphiques pour Windows (ABC Graphics Suite, Windows Draw 4.0, Designer, Picture Publisher, Instant 3D, Simply 3D, Visual Reality...), vient de signer un accord de licence avec Microsoft pour intégrer le langage Visual Basic Edition Application dans ses applications. A terme, les programmeurs en Visual Basic pourront utiliser les logiciels graphiques de Micrografx, comme de simples composants graphiques programmables.

Pentium Multimédia

Le Pentium Multimédia (MMX) n'est pas encore disponible. Mais les principaux éditeurs ont annoncé des versions MMX de leurs produits. A quand le Visual Basic MMX, le Visual C MMX et le Delphi MMX ? Le système graphique DirectX MMX pour Windows 95 est annoncé pour la fin de l'année.

Internet

Borland a annoncé la sortie imminente d'une nouvelle version de Delphi capable de générer des applications et des serveurs Internet. Avec le Borland C++ 5.2 qui peut compiler des applications Java, Borland est en train de devenir un acteur important du monde Internet. IBM a acheté une licence Java. L'ex-géant de l'informatique serait-il en train de rattraper le train en marche ? Va-t-il combiner cela avec le système d'exploitation 7.0 acheté à Apple ?
La société Netscape n'apprécie pas trop que Microsoft fournisse gratuitement la dernière version de Microsoft Internet Explorer. Elle a porté plainte devant le gouvernement américain pour pratique commerciale illicite. La justice américaine a poussé un soupir et ajouté quelques pages à l'épais dossier Microsoft qui se remplit lentement tous les jours. Voici quelques années, la commission américaine anti-monopole a forcé les Laboratoires Bell à diviser ses activités en plusieurs sociétés indépendantes. Est-ce un avenir possible pour Microsoft ?

Windows

Microsoft est sur le point de lancer une grosse campagne de vente pour Windows NT 4.0. Notre bon vieux Windows 95 est-il déjà mort, alors que de nombreux utilisateurs se ser-vent encore de Windows 3.1 ? Rappelons que les logiciels Windows 16 bits ne pourront pas fonctionner avec Windows NT 4.0. Seules les applications 32 bits développées avec Windows 95 seront compatibles avec le nouveau système d'exploitation. En fait, le véritable problème est de savoir quelle est la configuration machine nécessaire pour le nouveau système (32 Mo, Pentium 133 ou plus ?).

vos questions, nos réponses...

Nous avons besoin de vos conseils et de vos questions afin de vous offrir le meilleur magazine possible. Quels que soient vos problèmes, n'hésitez pas à nous écrire. Ces pages sont ouvertes à tous. Et n'oubliez jamais que ce magazine est le vôtre !

Qbasic

J'ai acheté le numéro 6 de votre revue pour les articles en Qbasic. C'est intéressant, mais j'aurais aimé que vous expliquiez le fonctionnement de la procédure LoadBmp car c'est le coeur de l'article. Les paramètres les plus déroutants appellent peut-être un brin de commentaire. Ainsi, dans la ligne «rien=ASC(Input$(1,fic))/4», je ne comprends pas le rôle de «/4». C'est la même chose pour les instructions «seek fic, 54» et «seek fic, 1079». Que signifient ces chiffres ? De plus, j'ai eu un message d'erreur «appel de fonction non permis» en utilisant votre routine. Pourquoi ?

L'article Qbasic de PC FUTE n°6 traitait des instructions graphiques GET et PUT qui permettent de stocker des images en mémoire et de les rappeler à volonté et de l'affichage d'images en ombres chinoises. Plusieurs programmes d'exemples utilisent la procédure LoadBmp pour charger des images BMP 320x200 à partir du disque, mais le but de l'article n'est pas le chargement d'une image BMP. LoadBmp est une procédure standard faisant partie de la bibliothèque classique d'un programmeur en Qbasic, avec des performances plutôt limitées. C'est pourquoi nous n'avons pas étudié la question en détail.

Pour répondre à votre question, l'instruction «rien=ASC(Input$(1,fic))/4)» charge un octet à partir d'un fichier. Il s'agit de l'intensité lumineuse d'une couleur, codée sur 256 valeurs différentes (0= pas de lumière, 255 = lumière intense). La palette de couleurs VGA utilisée par le Qbasic gère seulement 64 niveaux de luminosité (0 = pas de lumière, 63 = lumière intense). La routine de chargement doit donc convertir le codage sur 256 valeurs en un codage sur 64 valeurs. La manière la plus simple est de diviser l'indice de lumière par 4 (256/64=4). Cela fonctionne parfaitement !

Vous vous demandez à quoi servent les instructions «seek fic,54» et «seek fic,1079». Elles permettent au programme d'accéder à différents endroits du fichier contenant l'image BMP, sans lire tout les octets intermédiaires. L'instruction «seek fic,54» positionne le programme au début de la zone contenant la palette de couleurs de l'image et l'instruction «seek fic,1079» place le programme au début de la zone contenant l'image elle-même !

Vous trouverez dans ce magazine un article expliquant comment charger une image BMP dans un programme en Langage C. Ce n'est pas du Qbasic, mais le principe de chargement est le même, quel que soit le langage utilisé !
La routine LoadBmp est très lente à charger une image BMP car le Qbasic manque d'instructions permettant d'accéder rapidement aux fichiers et de manipuler la mémoire. Nous verrons bientôt comment remédier au problème et charger une image BMP très rapidement en Qbasic !

Vous avez eu un message d'erreur «appel de fonction non permis» en tentant de charger une image inexistante. C'est l'instruction Input$(1,fic) qui a généré l'erreur quand le Qbasic a tenté de lire un octet en provenance d'un fichier inexistant ou impossible à ouvrir, à cause d'un problème disque. Pour éviter cela, il faut vérifier si un fichier existe effectivement avant de l'ouvrir. L'article «Qbasic - Trucs et astuces» vous explique comment faire cela dans vos programmes.

Synthèse d'images

J'ai remarqué (Ô Horreur !!) que vous avez supprimé la rubrique «Initiation Persistence of Vision» dans la nouvelle formule de votre magazine. M'étant remis de mes émotions, je vous écris principalement pour cette raison. Ayant acquis POV avec le numéro 2 de PC GENIUS, je suivais cette rubrique avec assiduité. C'est pour cela que je prends la plume, ou plutôt le clavier, pour vous convaincre de remettre cette rubrique «en circulation».

Félicitations ! Vous avez réussi votre coup et notre rubrique «Synthèse d'image» va revenir dès le prochain numéro. Mais l'approche sera différente. Plutôt que d'expliquer des notions théoriques, nous préférons privilégier le côté «pratique» de l'image de synthèse.

Les thèmes traités seront le dessin de boutons 3D, la génération de polices de caractères en relief et le dessin d'éléments graphiques (logos, titres, animations, etc.) utilisables dans les programmes.

DOS

En voyant l'évolution du DOS et de Windows, je ne peux m'empêcher de me poser quelques questions. Pourquoi a-t-on fait évoluer Windows en 32 bits, alors que le DOS reste désespérément en 16 bits ? Je pense pourtant qu'une version 32 bits du DOS aurait eu un grand succès auprès des programmeurs et des utilisateurs !

Il est difficile de se mettre dans la peau de ceux qui décident de l'évolution de l'informatique, en fonction de critères complexes faisant intervenir un peu de technique, mais surtout de considérations commerciales et de luttes acharnés entre corporations géantes. C'est pourquoi nous n'avons pas eu de DOS 32 bits, mais Windows 3.1, Windows 95 et bientôt Windows NT 4.0 ! C'est dommage pour les utilisateurs et tant mieux pour les industriels qui peuvent vendre de nouvelles machines et de nouveaux systèmes. Nous sommes pris dans une spirale infernale où le credo est «Toujours plus!» à la place de «Toujours mieux !». C'est dommage, mais c'est comme cela !

Pourquoi y a-t-il tant de problèmes avec les cartes graphiques et sonores sous DOS ? Pourquoi n'y a-t-il pas de systèmes de drivers standard pour le DOS ? Cela existe bien pour Windows, alors que c'est un système bien plus complexe que le DOS ! Cela faciliterait considérablement la vie des pauvres programmeurs comme moi ! Si j'ai bien compris vos articles, le système DirectX pour Windows est essentiellement une série de drivers très rapides, adaptés à chaque carte graphique et sonore. Est-il possible d'utiliser ces drivers avec le DOS ?

Il n'existe pas de drivers graphiques et sonores universels sous DOS, parce que personne n'a fait l'effort de concevoir et d'imposer un système standard. Chaque constructeur suit ses propres idées et ne parle pas avec ces concurrents. Les seuls standards existants sur le marché sont des produits ayant tellement bien marché que tout le monde les a copiés (carte graphique VGA d'IBM et carte sonore Sound Blaster de Créative Labs). La seule tentative vraiment sérieuse pour créer un standard graphique a été la norme VESA. Mais les fabricants de cartes vidéo ont interprété d'une manière plutôt fantaisiste les spécifications graphiques VESA. Le seul mode graphique commun à toutes les cartes graphiques VESA est le mode 640x480 en 256 couleurs.

Il est dommage que Microsoft n'ait pas tenté d'imposer un système de drivers DOS standard. C'est la seule société ayant la puissance commerciale et le leadership technologique suffisant pour réussir ce tour de force. Mais les stratèges de Microsoft ont préféré parier sur Windows et laisser mourir le DOS par abandon. C'est pourquoi il n'existe pas de systèmes de drivers standard pour le DOS. Les drivers graphiques et sonores de DirectX ne peuvent pas être utilisés sous DOS car ils ont été conçus spécifiquement pour Windows 95.

Comment le Bios fait-il pour reconnaître nombre de matériels et rester compatible avec ?

Le Bios est totalement incapable de reconnaître quoi que ce soit ! Les seules cartes compatibles avec lui sont celles qui contiennent leurs propres extensions Bios. C'est le cas avec certains disques durs SCSI et les cartes graphiques VGA. La plupart des fonctions Bios de ces cartes graphiques ne sont pas vraiment utilisables par les programmes, car elles correspondent à des normes graphiques spécifiques à chaque constructeur. Les seuls cas de fonctions Bios (presque) totalement compatibles sont celles de la souris, car tous les constructeurs ont copié les spécifications de la première souris Microsoft.

Visual Basic

Je vous écris pour vous dire que je suis bien content que vous vous aperceviez enfin que tout le monde ne dispose pas de Visual Basic 4.0, ni de Windows 95. Nous avons eu déjà du mal à comprendre notre Windows 3.1 et je suis sûr qu'il existe encore beaucoup de gens qui utilisent le Visual Basic 2.0 sur des 386 SX. Vos articles sur l'initiation au Visual Basic sont très bien, mais à raison de 1 ou 2 instructions, propriétés ou astuces par numéro, nous aurons fini de faire le tour du VB 3.0 et 4.0 vers l'an 2387.

Vos calculs sont exacts, mais nous n'avons pas l'intention de présenter toutes les instructions du Visual Basic. C'est le travail des éditeurs de livres et ils le font bien, vu le poids des derniers ouvrages sur le VB. Notre but n'est pas de concurrencer ces superpuissances de l'édition informatique, mais de vous aider dans vos problèmes quotidiens en VB, d'attirer votre attention sur certaines instructions intéressantes et de répondre à vos questions.

Nous sommes conscients que tout le monde ne possède pas Windows 95, ni le Visual Basic 4.0. C'est pourquoi nous allons nous concentrer sur le VB 3.0, sans toutefois oublier de parler du VB 4.0, chaque fois que ce sera nécessaire ! En général, la transition est très rapide entre deux versions d'un même langage. Mais ce n'est pas le cas avec le VB 4.0, où il faut également changer de système d'exploitation et passer à Windows 95, ce qui nécessite un PC musclé ! Le même problème se pose avec Delphi 2.0 qui ne fonctionne qu'avec Windows 95. C'est pourquoi nous allons parler simultanément de Delphi 1.0 (Windows 3.1) et de Delphi 2.0 afin de ne léser personne.

Windows

Quand on voit la vitesse des applications en «vrai» multitâche comme Unix et OS/2, on se pose des questions sur les raisons qui font sortir les logiciels Microsoft avec du retard. Pourquoi Windows 95 est-il moins performant que ces autres systèmes d'exploitation ?

Les systèmes d'exploitation dont vous parlez ont été écrits en code 32 bits, de manière à utiliser au mieux les performances des PC modernes. Ils fonctionnent en multitâche préemptif, c'est-à-dire que le temps d'exécution est réparti équitablement entre tous les programmes actifs. C'est le système d'exploitation qui gère cette attribution de temps. Cette technique nécessite que le système d'exploitation soit capable de gérer les conflits de périphériques. En effet, il peut se produire un problème si deux programmes tentent d'accéder simultanément au même fichier ou d'écrire en même temps sur l'écran. Windows 3.1 n'est pas un vrai système d'exploitation, son fonction-nement repose sur le DOS qui n'a pas été prévu pour gérer les conflits de périphériques. C'est pourquoi il utilise une technique de multitâche non-préemptif. Cela veut dire que ce sont les programmes qui gèrent eux-mêmes le multitâche. Chaque application prend le contrôle du système pendant un certain temps et décide elle-même du moment où elle signale à Windows 3.1 qu'il peut donner le contrôle à une autre application. Cela évite les conflits de périphériques, les programmes ne redonnant la main à Windows que lorsque les accès aux périphériques sont terminés. C'est une technique dangereuse, car un programme mal écrit peut perturber le fonctionnement du multitâche et bloquer l'ensemble du système.

Si Windows 95 était un vrai système d'exploitation 32 bits avec un multitâche préemptif, il ne pourrait pas exécuter les programmes écrits pour Windows 3.1. C'est pourquoi Windows 95 est un système hybride qui peut exécuter des programmes 32 bits en mode multitâche préemptif et des programmes 16 bits (Windows 3.1) en multitâche non-préemptif. Cette compatibilité fait que certaines parties de Windows 95 sont encore écrites en code 16 bits. Tout cela limite les performances de Windows 95, par rapport à de vrais systèmes d'exploitation écrits directement en code 32 bits avec un multitâche préemptif !

Les premiers PC ont été fabriqués avec le micro-processeur 16 bits Intel 8088 dont l'architecture interne était très mal conçue. Mais le DOS et tous les programmes 16 bits ont été conçus pour fonctionner avec ce microprocesseur. Avec le premier microprocesseur 386, les ingénieurs d'Intel ont fait un grand pas en avant. C'est un circuit qui intègre sur la même puce un micro-processeur 32 bits performant et un vieux 8088 16 bits. Mais le logiciel évolue moins vite que le matériel ! Le DOS et Windows continuant à fonctionner en mode 16 bits, les programmes sont restés en code 16 bits ! En réalité, Windows 95 est un système de transition entre Windows 3.1 et le futur système d'exploitation 32 bits Windows NT 4.0. C'est pourquoi ses performances sont inférieures à celles de vrais systèmes d'exploitation 32 bits ! Il faut encore attendre quelques mois pour savoir si Windows NT 4.0 tiendra ses promesses.

Pourquoi dit-on que Windows 95 est Plug and Play, alors que tout ce qu'il sait faire c'est utiliser des drivers existants comme sous Dos ?

Windows 95 est Plug and Play (Brancher et Utiliser), c'est-à-dire qu'il peut reconnaître et utiliser automatiquement n'importe quel périphérique. Mais pour que cela fonctionne, il faut que tous les périphériques soient également Plug and Play, ce qui est loin d'être le cas ! Une carte sonore classique pose les mêmes problèmes d'installation sous Windows 95 que sous DOS ! Il faudra encore attendre un ou deux ans pour que toutes les cartes d'extension soient à la norme Plug and Play.

Divers

Vous avez expliqué comment programmer un casse brique en langage C. C'est stupide car il y a un exemple de ce type en Turbo Pascal. Si vous ne voulez plus parler du Turbo Pascal, dites-le franchement !

Le fait de parler de la programmation d'un casse brique en langage C serait effectivement un acte stupide si tous les programmeurs en langage C étaient des spécialistes du Turbo Pascal, ce qui est loin d'être le cas ! Ceci dit, nous allons arrêter le Turbo Pascal pour nous consacrer à Delphi, dont le véritable nom aurait dû être Visual Turbo Pascal pour Windows !

J'aimerais émettre quelques remarques concernant PC FUTE, sur le fait d'attaquer tous azimuts, c'est-à-dire de faire des articles sur le C, puis dans le même numéro sur Delphi et sur le Visual Basic. Ne croyez-vous pas que certains lecteurs ne vont pas savoir par où commencer ? Ils vont avoir tendance à se disperser et à apprendre dans toutes les directions, sans pour autant évoluer correctement ? Ne faudrait-il pas donner une marche à suivre au débutant ? N'allez vous pas faire en sorte de laisser sur la route certains de vos lecteurs qui n'auront pas pu suivre ? Ceci dit, je suis ravi de vos articles et je suis très content de votre orientation création de jeux et graphisme.

Le marché français de la presse informatique est encore trop petit pour que nous sortions une revue consacrée au Visual Basic, une revue sur le C, une revue sur Delphi et une revue pour les autres langages. C'est dommage, mais la réalité économique nous oblige à traiter de tous ces langages dans la même revue. Ceci dit, un article sur la programmation en Visual Basic ou en Qbasic peut apprendre quelque chose d'utile à un programmeur Delphi. Les langages sont différents, mais les principes sont les mêmes. Par exemple, ce magazine contient un article expliquant comment écrire un jeu de type Civilisation en Langage C. Les techniques exposées peuvent être reprises en Turbo Pascal, en Visual Basic, en Delphi et même en Qbasic !

L'apprentissage de la programmation est une démarche personnelle. Il n'y a pas de technique infaillible pour apprendre l'informatique. Si c'était le cas, elle serait déposée depuis longtemps. Pour s'en convaincre, il suffit de regarder les livres parlant d'informatique. Il n'existe aucun ouvrage universel capable d'amener un débutant au même niveau qu'un expert. De plus, nous n'avons pas envie d'écrire des articles sur des choses évidentes, abondamment expliquées dans les manuels techniques et les livres. Notre rôle est surtout d'attirer votre attention sur certains aspects élémentaires de la programmation, généralement oubliés par les auteurs qui écrivent les ouvrages informatiques. Nous ne sommes pas un guide qui vous indique la route en permanence, mais plutôt un éclaireur qui balise le chemin, en laissant des torches allumées aux endroits dangereux ! Chaque lecteur doit trouver son chemin, en tenant compte de nos indications de dangers !

La Rédaction

Nous avons besoin de vous !

Il est difficile de parler d'initiation à la programmation sans connaître vos désirs. Que pensez-vous de cette série d'articles ? Y a-t-il des choses qui ne sont pas claires ? Voulez-vous des explications sur un point particulier ? Nous avons besoin de vos avis pour savoir ce qui vous intéresse. N'hésitez pas à nous écrire. N'oubliez jamais que ce magazine est le vôtre !

Expédition
Les disquettes contenant vos créations doivent être envoyées avant le 15 octobre à l'adresse suivante :
Concours PC FUTE
24 bis, rue du Gabon
75012 PARIS

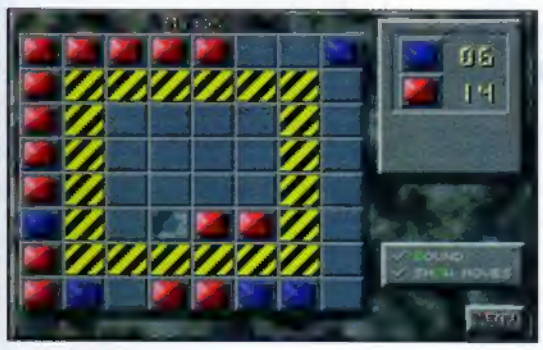
Un concours pour les créateurs

VOULEZ-VOUS DEVENIR UN CRÉATEUR DE JEUX VIDÉO ? AVEZ-VOUS ÉCRIT UN JEU GÉNIAL QUI DORT DANS UN TIROIR ? AVEZ-VOUS DES IDÉES ORIGINALES SUR LA MANIÈRE DE CRÉER UN JEU ? ETES-VOUS UN BON GRAPHISTE ? SI OUI, PARTICIPEZ À NOTRE CONCOURS ! LES MEILLEURES CRÉATIONS SERONT PUBLIÉES DANS NOTRE MAGAZINE.

Le concours annoncé dans PC FUTE n° 7 continue. Vous avez jusqu'au 15 octobre pour nous envoyer vos créations. Le thème est la création d'un jeu vidéo pour le *DOS* ou *Windows*. Tous les langages de programmation classiques peuvent être utilisés. Il y a un concours pour les programmeurs et un autre pour les graphistes. Les gagnants verront leurs créations publiées dans le magazine et recevront un chèque de 1 000 F à titre de droits d'auteur.

Les jeux

Le jeu ne doit pas tenir sur plus de 3 disquettes. Si le système de programmation utilisé nécessite un runtime, comme le fichier *VBRUN300.DLL* du *Visual Basic 3.0*, celui-ci peut être fourni sur une seconde disquette. Le listing source doit nécessairement être présent. Il doit comporter un maximum de commentaires. Un petit fichier texte portant le nom d'*INTRO.TXT* doit présenter sommairement le logiciel. La documentation du jeu doit se trouver dans le fichier texte *DOC.TXT*. Si vous avez le courage de taper un texte parlant de la manière dont vous avez conçu le jeu et comment il fonctionne, il doit se trouver dans le fichier *INFOS.TXT*.
Les critères de sélection seront l'originalité, le plaisir de jeu, l'esthétisme et la qualité technique. Vous pouvez nous envoyer autant de programmes que vous voulez. Aussi, n'hésitez pas à vider vos fonds de tiroirs ! Il y a peut-être une petite merveille qui dort dedans depuis longtemps ! Le thème n'est pas imposé. Vous pouvez nous envoyer aussi bien un jeu d'aventure, un jeu de logique, un jeu de simulation, un wargame, un jeu d'action, etc. L'intérêt du jeu compte plus que le thème.



▲ *Les débutants peuvent commencer par créer des jeux de réflexion. Cela ne nécessite pas des connaissances techniques très poussées et le résultat peut être intéressant, à condition d'avoir de bons graphismes.*

Les graphismes

Ce concours est réservé aux images pouvant servir à la réalisation d'un jeu. Ce peut être une image complète ou un assemblage de petits éléments graphiques (16x16 pixels pour les jeux *Dos* et 32x32 pour les jeux *Windows*). Par exemple, vous pouvez créer un jeu d'éléments de base permettant de dessiner une forêt, une jungle, un temple, un château fort, l'intérieur d'un vaisseau spatial, une grotte, etc.
Les images doivent être fournies au format graphique *BMP* 256 couleurs ou *GIF* 256 couleurs. Ce sont les formats graphiques les plus utilisées. La palette de couleurs doit être identique pour les éléments d'un même jeu. Les graphistes ont souvent du mal à se rappeler ce détail. Rien n'est plus crispant pour un programmeur que de s'apercevoir au moment de l'affichage qu'un sprite n'utilise pas la même palette que le reste des éléments graphiques !
Les graphismes peuvent êtres dessinés à la main ou créés avec un logiciel de raytracing de type *POV*. Ce n'est pas la technique de création qui compte, mais la beauté des images. Les graphistes professionnels utilisent les deux méthodes pour créer des jeux.

La Rédaction

Et le son ?
Certains membres de la Rédaction nous ont fait remarquer que le son est très important pour les jeux vidéo. Si vous avez créé des sons digitalisés ou des musiques originales, envoyez-les nous ! Si nous recevons suffisamment de créations sonores, nous organiserons un concours de sonorisation.

Inscrivez-vous dans notre annuaire

VOUS ÊTES UN PROGRAMMEUR CHERCHANT UN GRAPHISTE OU UN GRAPHISTE CHERCHANT UN PROGRAMMEUR ? VOUS ÊTES MEMBRE OU ANIMATEUR D'UN CLUB INFORMATIQUE ? VOUS CHERCHEZ DES CONTACTS OU DES PARTENAIRES POUR DÉVELOPPER UN PROJET INTÉRESSANT ? UNE SEULE SOLUTION, INSCRIVEZ-VOUS À NOTRE WHO'YOU INFORMATIQUE !

Notre Adresse
PC FUTE
Annuaire
24 bis, rue du Gabon
75012 PARIS

Un lecteur nous a suggéré de créer un annuaire informatique regroupant les adresses des personnes intéressées par la programmation. L'idée est intéressante car elle permet de faciliter l'échange d'informations et d'idées entre passionnés de programmation. Elle facilite aussi la constitution d'équipes pluridisciplinaires car il existe beaucoup de programmeurs, de graphistes et de musiciens cherchant des partenaires pour créer des logiciels.

L'opération concerne aussi les clubs d'informatique qui sont des endroits merveilleux pour rencontrer des gens intéressants, suivre des cours et travailler en commun sur de gros projets.

L'annuaire sera distribué sous la forme d'un fichier texte sur la disquette du mois. Cela ne prend pas de place dans le magazine et permet une consultation rapide. Nous la mettrons à jour à chaque nouvelle parution de PC FUTE.

L'inscription est gratuite. Il suffit de nous écrire au magazine. Vous pouvez également répondre au sondage. Le numéro de téléphone est optionnel afin d'éviter les plaisantins qui donnent les numéros de leurs amis pour leur faire une blague.

La Rédaction

ANNUAIRE PC FUTE

Nom .....

Adresse .....

.....

Téléphone .....

❑ **Programmeur professionnel** ❑ **Programmeur amateur**

❑ **Autre (précisez)** .....

❑ **Débutant** ❑ **Expérimenté**

Langages utilisés

.....

.....

Centres d'intérêt

.....

.....

Cherchez-vous des contacts ?

.....

.....

Vos commentaires

.....

.....

.....

UN LANGAGE DE PROGRAMMATION EST UN SYSTÈME QUI PERMET DE CRÉER UN PROGRAMME. MAIS CELA N'A RIEN À VOIR AVEC LE LANGAGE HUMAIN ! LA RÉALISATION D'UN PROGRAMME RESSEMBLE PLUS À L'ÉLABORATION D'UNE RECETTE DE CUISINE QU'À UNE CONVERSATION ENTRE DEUX INTERLOCUTEURS !

Qu'est-ce qu'un langage de programmation ?

Le cerveau humain est un système très sophistiqué qui apprend à faire des choses par apprentissage et imitation. Il se compose d'un gigantesque assemblage de neurones qui peuvent s'assembler par réseaux d'analyse spécialisés dans certains domaines. Par exemple, il existe des assemblages de neurones spécialisés dans la détection des objets mobiles, l'analyse des contours, la reconnaissance des visages, l'identification du goût, le contrôle des doigts, etc. Tout cela fonctionne en même temps, d'une manière que les scientifiques commencent seulement à comprendre. C'est très complexe car les neurones se sont organisés eux-mêmes en réponse aux perceptions reçues pendant les premières années de l'enfance. C'est une sorte de fourmilière, où chaque élément agit indépendamment des autres et dont la combinaison de toutes ces actions forme un tout cohérent.
Les ordinateurs fonctionnent différemment. Nous ne savons pas encore comment créer des machines fonctionnant de la même manière que le cerveau humain. Tout ce que nous savons faire est de créer des sortes de grosses calculatrices dont l'intelligence ne dépasse pas celle d'un grille-pain ou d'un ouvre-boîtes. La seule chose que ces machines primitives sont capables de faire est d'exécuter mécaniquement des instructions simples. C'est l'équivalent moderne des orgues de barbarie qui jouent de la musique écrite sur une bande de papier perforée.

On peut comparer un ordinateur à une sorte de robot cuisinier exécutant automatiquement les différentes phases d'une recette de cuisine. Notre robot cuisinier peut exécuter par lui-même des ordres simples comme prendre x oeufs, les battre pendant x minutes, prendre x grammes de beurre, allumer le four, mettre x litres de lait dans un saladier, couper des tomates, transférer le contenu d'un plat dans un autre, découper un morceau de viande en x parts égales, préparer un bouillon, mettre un plat dans le four, attendre x minutes, retirer le plat du four, etc. Pour confectionner une recette particulière, il faut indiquer au robot la séquence précise d'ordres qu'il doit exécuter pour réaliser le plat. Cela donne :

```
- PRENDRE 3 OEUFS
- PRENDRE 100 GRAMMES DE FARINE
- PRENDRE 0,5 LITRE DE LAIT
- PRENDRE UNE PINCEE DE SEL
- PRENDRE UN BOL
- CASSER LES OEUFS DANS LE BOL
- MELANGER LE CONTENU DU BOL PENDANT
1 MINUTE
- METTRE LA FARINE DANS LE BOL
- METTRE LE SEL DANS LE BOL
- MELANGER LE CONTENU DU BOL PENDANT
3 MINUTES
- METTRE LE LAIT DANS LE BOL
- MELANGER LE CONTENU DU BOL PENDANT
3 MINUTES
```

Quels langages pour débuter ?

Le monde du PC est partagé entre l'ancien système Dos et Windows. C'est la même chose pour les langages de programmation, bien que les éditeurs ne commercialisent plus de langages DOS depuis quelques années.
Le langage de base pour le DOS est le Qbasic. C'est un langage simple, qui de plus est offert gratuitement par Microsoft avec le DOS. C'est un avantage non négligeable pour les gens qui ont un budget informatique serré.
Il n'existe pas vraiment de langages d'initiation pour Windows.
Les éditeurs ne s'intéressent pas à ce créneau et préfèrent commercialiser des langages professionnels, puissants et surtout coûteux ! Néanmoins, il existe deux langages très intéressants pour les débutants : le Visual Basic et Delphi. Ils sont simples à utiliser et permettent de créer des applications performantes.

Pour définir avec précision une recette complète, il faudrait une bonne cinquantaine de lignes. Chaque instruction correspond à un ordre que peut exécuter le robot. Si la machine est incapable d'accomplir une tâche particulière, il faut la décomposer en tâches plus simples que le robot est capable d'exécuter. Prenons par exemple une recette pour faire une tasse de thé.

- PRENDRE 1 SACHET DE THE
- PRENDRE 0,3 LITRE D'EAU
- PRENDRE 3 SUCRES
- PRENDRE UN FOUR A MICRO-ONDE
- PRENDRE UNE TASSE A THE
- METTRE EAU DANS TASSE
- PLACER TASSE DANS MICRO-ONDE
- REGLAGE DU MICRO-ONDE SUR 3 MINUTES
- ALLUMER LE MICRO-ONDE
- ATTENDRE 3 MINUTES
- COUPER LE MICRO-ONDE
- RETIRER TASSE DU MICRO-ONDE
- METTRE THE DANS TASSE
- ATTENDRE 2 MINUTES
- RETIRER SACHET DE LA TASSE
- METTRE LES SUCRES DANS LA TASSE
- MELANGER CONTENU TASSE

Il est possible que le robot ne connaisse pas la fonction «METTRE LES SUCRES DANS LA TASSE». Après tout, la plupart des gens ne mettent qu'un sucre dans le thé. Dans ce cas, le robot ne connaît que l'instruction «METTRE UN SUCRE DANS LA TASSE». Pour mettre les 3 sucres dans la tasse, il faut répéter 3 fois la même instruction.

- METTRE UN SUCRE DANS LA TASSE
- METTRE UN SUCRE DANS LA TASSE
- METTRE UN SUCRE DANS LA TASSE

Notre robot est capable d'exécuter n'importe quelle recette, à condition qu'un cuisinier humain ait le courage de convertir ses propres recettes en une série d'ordres exécutables par le cuisiner robot. La conversion d'une recette en ordres robotiques peut prendre beaucoup de temps. Il faut d'abord analyser la manière dont le cuisinier prépare sa propre recette. Ce n'est pas évident car l'esprit humain a tendance à faire des choses sans vraiment y porter attention. Il suffit parfois d'un déséquilibre subtil entre deux épices pour rater un plat délicat.

Une fois que notre cuisinier sait exactement comment préparer sa recette, il doit définir les différentes phases de la préparation dans le langage robotique. Ce n'est pas toujours évident et il faut parfois ruser comme dans le cas des 3 sucres ! Une fois la recette prête et entrée dans la mémoire du robot, il faut la tester. La seule manière est de lancer l'exécution du robot et de voir ce qui se passe. Si le plat n'est pas réussi, il faut examiner ce que fait le robot étape par étape et voir l'endroit où se produit le problème. Par définition, le robot ne peut pas faire d'erreurs. Il se contente d'exécuter fidèlement les ordres. Si le plat n'est pas bon, le problème vient nécessairement de la recette.

A partir du moment où vous avez un robot cuisinier, vous pouvez créer toutes les recettes que vous voulez. Le robot n'hésitera pas à préparer 1 000 plats de suite, à condition que tous les ingrédients soient disponibles. Les systèmes robotiques et informatiques exécutent tous les ordres sans jamais discuter, puisqu'ils n'ont aucune intelligence. C'est un gros avantage par rapport à un cuisinier humain. C'est aussi un inconvénient car ces systèmes n'ont aucun sens critique et peuvent créer les pires catastrophes. Un cuisinier humain s'arrête de travailler quand le chat monte sur la table. Ce n'est pas le cas du robot qui continue tranquillement à couper la viande, même si le chat s'approche de trop de près de la planche à découper !

Qu'est-ce qu'un langage de programmation ?

Un langage de programmation est l'ensemble des instructions que peut exécuter un ordinateur. Dans le cas du robot, le langage de programmation est l'ensemble des ordres que le robot peut accomplir. L'ordinateur est identique au robot, c'est-à-dire qu'il n'est capable d'exécuter qu'une action à la fois. Un programme est similaire à une recette de cuisine, à ceci près que les commandes de base d'un langage ne ressemblent pas à des ordres culinaires. Ce sont plutôt des commandes du genre «AFFICHER INFORMATION SUR ECRAN», «LIRE LE CLAVIER», «ADDITIONNER DEUX CHIFFRES», «TRACER UN CERCLE SUR L'ECRAN», etc. C'est ce qu'on appelle les instructions du langage.

Les programmes sont plus complexes que les recettes de cuisine et ont besoin de stocker des informations. C'est pourquoi les langages de programmation sont capables de stocker des informations en mémoire. C'est

```
Module1
Objet: (Général)                    Proc: OpenFile
Sub OpenFile(Filename As String)
Dim F As Integer
    If "Editeur de texte : " + Filename = frmEditor.Caption Then
        Exit Sub
    Else
        On Error GoTo errhandler
            F = FreeFile
            Open Filename For Input As F
            frmEditor!txtEdit.Text = Input(LOF(F), F)
            Close F
            UpdateMenu
            frmEditor.Caption = "Editeur de texte : " + Filename
            Exit Sub
    End If
errhandler:
        MsgBox "Erreur lors d'une tentative d'ouverture de fichier, réessayez.
        Close F
        Exit Sub
End Sub
```

▲ *Voici un exemple de routine conçue pour exécuter une tâche bien précise.*

un aspect fondamental de la programmation car les ordinateurs sont surtout des machines à traiter l'information. Pour cela, les ordinateurs utilisent des variables. Ce sont de petits morceaux de la mémoire qui ne servent qu'à stocker des données numériques. Ces variables sont identifiées par des noms, afin de faciliter leur utilisation. Le programme suivant vous montre comment il est possible d'additionner deux nombres et d'afficher le résultat sur l'écran.

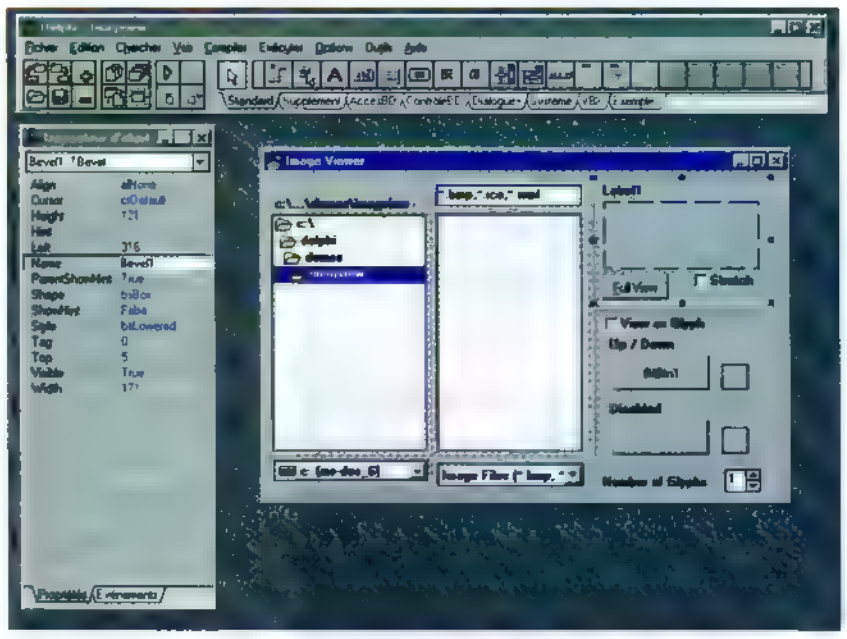
```
- CREATION VARIABLE A
- CREATION VARIABLE B
- CREATION VARIABLE C
- A = 18
- B = 23
- C = A + B
- AFFICHER VARIABLE C SUR ECRAN
```

Ce programme est théorique, mais il est possible d'utiliser un vrai langage de programmation pour faire la même chose. Nous avons choisi pour cela le *Qbasic*, un langage simple d'emploi. Voici la version *Qbasic* de notre petit programme :

```
DIM A, B, C

A = 18
B = 23
C = A + B
PRINT C
```

Les structures de contrôle

Une recette de cuisine est un processus linéaire, dont les étapes s'enchaînent les unes après les autres. Ce n'est pas le cas des programmes informatiques dont le fonctionnement doit être plus souple que l'exécution d'une recette de cuisine. Nos «recettes informatiques» doivent s'adapter aux impondérables. C'est pourquoi les langages de programmation sont équipés de structures de contrôle permettant de surveiller et de modifier le déroulement des programmes.



▲ *Delphi est un excellent langage de programmation pour créer des logiciels au look Windows.*

La structure de test peut exécuter une séquence d'instructions particulière si une situation précise se présente. L'exemple suivant est un programme qui lit une information à partir du clavier et arrête le déroulement du programme si le clavier fournit une certaine valeur.

```
C = LECTURE CLAVIER
SI C = 27 ALORS FIN PROGRAMME
AFFICHER C SUR ECRAN
```

Pour que cela fonctionne, il faut que le langage de programmation ait une instruction de lecture du clavier, une instruction qui stoppe le programme et une instruction qui affiche le contenu d'une variable sur l'écran.

La structure de répétition est très importante. Elle permet d'exécuter plusieurs fois la même action. Par exemple, vous voulez afficher 5 fois de suite le message «BRAVO» sur l'écran de votre machine. Vous pouvez écrire 5 fois la même instruction dans le programme, de la manière suivante :

```
- AFFICHER «BRAVO» SUR ECRAN
- AFFICHER «BRAVO» SUR ECRAN
- AFFICHER «BRAVO» SUR ECRAN
- AFFICHER «BRAVO» SUR ECRAN
- AFFICHER «BRAVO» SUR ECRAN
```

Ce n'est pas très élégant. Il est préférable de dire à la machine d'exécuter 5 fois l'instruction d'affichage.

```
- REPETER 5 FOIS:
    AFFICHER «BRAVO» SUR ECRAN
```

C'est plus propre et plus souple. En utilisant la même instruction, il est possible d'afficher le message 10 fois ou même 100 fois. Il suffit juste de changer un chiffre dans le programme :

```
- REPETER 100 FOIS:
    AFFICHER «BRAVO» SUR ECRAN
```

Les langages de programmation possèdent de nombreuses structures de contrôle. Certaines sont des tests (égalité, supériorité, infériorité, etc.). Les autres sont des structures répétitives qui exécutent la même opération un certain nombre de fois, ou jusqu'à ce qu'une condition précise se produise. Ces instructions sont vitales pour la création des programmes car elles permettent de modifier le comportement des programmes en fonction des événements extérieurs.

Les routines

La création d'une recette passe souvent par l'utilisation d'autres recettes. Par exemple, pour préparer une salade de riz, il faut des oeufs durs et de la mayonnaise. La préparation des oeufs et de la mayonnaise sont deux recettes spécifiques. Le cuisiner doit commencer par préparer ces deux recettes avant de s'occuper de la salade. C'est la même chose en informatique où certaines tâches peuvent être ▶

considérées comme de petites recettes, indépendantes du programme principal.

Les langages de programmation sont conçus de manière à ce que le programmeur puisse créer ses recettes de base. C'est ce qu'on appelle les routines. Une routine est un ensemble d'instructions réalisant une tâche précise. Il peut y avoir une routine pour dessiner un objet sur l'écran, une routine pour sauver une information dans un fichier, une routine pour trier une liste de noms par ordre alphabétique, une routine pour émettre un son, etc. Avant d'écrire un programme, il faut posséder une collection de routines de base.

Système de développement

Comme nous l'avons vu, un langage de programmation se compose d'un certain nombre d'instructions. Mais il faut également des outils logiciels pour entrer le programme en mémoire et le tester. C'est ce qu'on appelle un système de développement. Un système de développement se compose d'un éditeur, d'un compilateur et d'un débugeur. L'éditeur est une sorte de traitement de textes simplifié qui sert à taper et à modifier les instructions du programme. C'est avec ce système que vous entrez vos recettes de cuisine dans la mémoire de la machine.

Le compilateur remplit une tâche plus subtile. En effet, le seul langage que l'ordinateur soit capable de comprendre est le langage machine. C'est un affreux jargon technique très difficile à comprendre et à utiliser. L'ordinateur est dans la même situation qu'un cuisinier étranger très compétent qui travaille à Paris sans rien connaître du français. Il faut un interprète pour lui donner des instructions. Cet interprète, c'est le compilateur. Il s'agit d'un logiciel très sophistiqué qui convertit un programme écrit dans un langage informatique classique en un programme en langage machine, compréhensible par l'ordinateur.

Le débugeur est un système qui permet d'exécuter un programme en «mode ralenti» et d'examiner ce qui se passe à l'intérieur. C'est une sorte de «traceur» qui donne la possibilité de comprendre ce que fait le programme et de voir ce qui arrive aux variables. Il autorise l'analyse en détail du fonctionnement d'un programme qui fonctionne mal et d'identifier la cause du problème. C'est un instrument précieux car il est rare qu'un programme fonctionne correctement du premier coup. Les programmeurs passent généralement plusieurs heures à bricoler dans les entrailles de leurs créations avant d'obtenir un fonctionnement correct !

Les systèmes de developpement Windows

Windows est une excellente interface graphique dont la vocation est de simplifier l'utilisation du PC. Mais cette simplicité d'emploi cache une grande complexité interne, ce qui ne facilite pas vraiment la programmation ! C'est pourquoi les langages de programmation *Windows* fonctionnent de manière graphique. Le programmeur dispose d'une collection d'objets graphiques qu'il place sur l'écran pour dessiner l'interface graphique de son programme. Il écrit ensuite les instructions qui doivent se produire lorsque l'utilisateur clique sur tel ou tel objet graphique. C'est un mélange entre le légo et les recettes de cuisine.

Les différents langages

Il existe de nombreux langages de programmation qui fonctionnent tous avec les mêmes principes de base. La différence se fait sur les instructions de base, les structures de contrôle, la manière dont les variables sont gérées et la vocation du langage. Un langage conçu pour écrire des programmes de gestion est différent d'un langage orienté vers les mathématiques. Les principaux langages actuels sont le *Basic*, le *Pascal* et le *C*.

Il existe des différences importantes entre les langages des différents éditeurs. Elles se font au niveau de l'environnement de développement, de l'efficacité du compilateur et du nombre de routines fournies en standard. Par exemple, le *Borland C*, le *Visual C*, le *Watcom C* et le *GNU C* sont très différents les uns des autres, alors que ce sont tous des compilateurs *C* !

Pour finir

Vous savez maintenant (en gros !) ce qu'est un langage de programmation. Il est temps de passer aux choses sérieuses. Précipitez-vous chez votre re-vendeur informatique, achetez le *Visual Basic* (la version standard à 800 F et non la version Pro à 3 000 F) et lancez-vous ! Et n'oubliez pas que les grands cuisiniers ont toujours commencé comme de simples marmitons !

Paul Leroy

Les jeux d'aventures

Les logiciels de gestion ne sont pas les seuls à manipuler de grosses quantités d'informations. C'est également le cas des jeux d'aventures. Cet article vous présente les principes de base nécessaires à leurs créations. Bienvenue dans le monde de la (petite) réalité virtuelle!

Les jeux d'aventures sont des logiciels où le joueur doit explorer une région, rencontrer des personnages étranges, résoudre des énigmes et trouver des objets. Ils ont été conçus comme des sortes de livres interactifs où le lecteur vit sa propre aventure et se crée une histoire personnelle.
De nombreux jeux d'aventures se déroulent dans des univers médiévaux, dominés par la magie et la force de l'épée. D'autres sont des aventures policières se déroulant de nos jours ou dans des périodes historiques proches. Et enfin, il existe des jeux de science-fiction plongeant les joueurs dans une ambiance futuriste, le plus souvent inspiré des films *Blade Runner*, *Star Trek* ou encore *Mad Max*.

Les ancêtres

Les premiers jeux d'aventures ont été écrits avant l'apparition des écrans graphiques. Ils fonctionnaient entièrement avec du texte, d'où leurs de jeux d'aventures textes, ou jeux d'aventures textuels. Le joueur communiquait avec le programme en tapant des ordres. Par exemple, «*ALLER NORD*» signifiait que le joueur voulait se déplacer vers le nord. De la même manière, «*EXAMINER ARMOIRE*» signifie que le joueur voulait jeter un coup d'œil approfondi sur une armoire. Le cœur de ces jeux était l'analyseur de syntaxe, dont la fonction était de comprendre les ordres du joueur. La plupart des analyseurs de syntaxe étaient sommaires et incapable de comprendre autre chose que des ordres très simples. Les meilleurs jeux d'aventures textes étaient anglais.

▲ *Un jeu d'aventure classique!*

Les premiers jeux graphiques

La généralisation du graphisme sur les premiers ordinateurs à provoqué une mutation des jeux d'aventures. Le principe de jeu est resté inchangé, mais certains textes étaient remplacés par des images ! Par exemple, l'ordinateur n'affichait plus «*Vous êtes devant une grande cascade. L'eau tombe en faisant un bruit sourd, masquant le chant des oiseaux et les bruits de la nature*». A la place, il y avait l'image d'une cascade ! Ceci dit, il n'y avait pas beaucoup d'images dans ces jeux à cause des capacités limités de stockage des disquettes et des performances limités des cartes graphiques de l'époque.

Les jeux d'aventures souris

Le problème le plus crucial de l'informatique est celui de l'interface utilisateur. Comment donner rapidement des ordres à l'ordinateur et recevoir des informations en retour ? Le système *clavier/écran*, ultime avatar des machines à écrire et des sys-

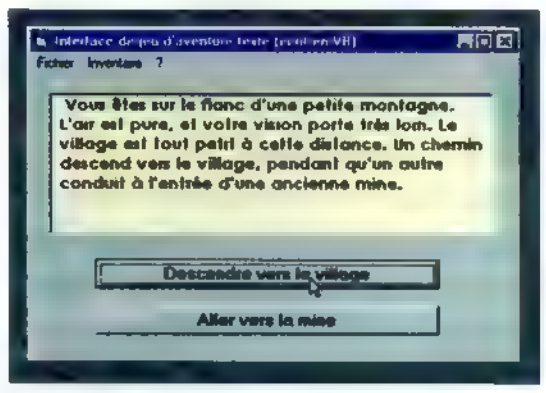
tèmes de dactylographie, n'est pas très efficace. La véritable révolution informatique à été provoqué par l'apparition de la souris et des écrans graphiques. Avec une souris, il est possible de créer des interfaces graphiques faciles à utiliser, y compris par des non-spécialistes ! La souris a sonné le glas des analyseurs de syntaxes et des jeux d'aventures difficiles à utiliser. Le joueur n'a plus à taper un ordre du type «*ALLER SUD*» ou «*ENTRER MAISON*» pour se déplacer. Il lui suffit de placer la souris sur l'endroit voulue et de cliquer. En règle générale, le curseur souris change de forme pour indiquer les actions possibles. Par exemple, la souris prend la forme d'une main quand elle se trouve au-dessus d'un objet qu'il est possible de prendre. C'est le principe utilisé par l'excellent jeu de science-fiction *BAT*.

Les jeux d'aventures cinématiques

Le progrès technique aidant, les ordinateurs sont devenus de plus en plus puissants. C'est alors que certains concepteurs de jeux d'aventures ont renoncés aux techniques traditionnelles pour créer les jeux d'aventures cinématiques. Ce sont des jeux où le décor est dessiné en 3D, et dans lequel le joueur dirige les actions d'un petit personnage. L'affichage est orienté de telle manière que le joueur surplombe le décor, et voit tout ce qui se passe. Par exemple, une scène peut se passer sur la place d'un village. Le joueur peut se déplacer pour examiner le contenu du puits, entrer dans la taverne, aller voir le maréchal-ferrant, discuter avec les habitants et même quitter le village. Les actions peuvent être commandées à la souris ou avec les flèches du clavier. Le jeu français *LBA* est l'un des meilleurs jeux cinématiques existants. Il doit son succès au principe de jeu, et surtout à la beauté de ses graphismes *SVGA* (640x480 en 256 couleurs).

Création d'un petit jeu d'aventure

Il est facile de programmer un petit jeu d'aventure, à condition d'être modeste et de ne pas tenter d'imiter les superproductions actuelles. Vous pouvez utiliser n'importe quel langage classique, comme le *Qbasic* ou le *Langage C*. Ceci dit, la tâche sera plus facile si vous employez un langage sophistiqué comme le *Visual Basic* ou *Delphi*.



▲ *Un système de déplacement simple, écrit en VB!*

Nous allons créer un jeu d'aventure très simple. Cela élimine les analyseurs de syntaxes et les jeux cinématiques. Le logiciel va être entièrement textuel, car tout le monde ne connaît pas nécessairement un graphiste prêt à dessiner une centaine d'images, juste pour faire un essai ! Toutes les commandes se feront à la souris.

Gestion de l'espace de jeu

Tous les jeux d'aventures sont constitués par une série d'endroits particuliers. Le jeu d'aventure médiéval type comprend une clairière, un chemin, une grotte, une maison dans les bois, une place de village, une taverne, une forge, l'échoppe d'un marchand, un entrepôt, la boutique d'une voyante, un poste de garde, un souterrain partant de la taverne, etc.. Chaque lieu peut être décrit par un texte. Les descriptions des lieux peuvent être stockées dans un tableau alphanumérique. C'est une structure de données qui se retrouve dans tous les langages modernes. Cela permet de donner un numéro d'identification à chaque lieu. Par exemple, la description du lieu 4 se trouve dans le tableau *Lieu(4)*.

```
' Qbasic
DIM shared lieu$(100)

' Visual Basic
Global lieu$(100)

// Langage C
char *Lieu[100];

(* Delphi *)
lieu : array [1..100] as string;
```

Le stockage des textes en mémoire peut se faire de différentes manières. Par exemple, le plus simple en *Qbasic* est d'utiliser l'instruction *DATA* pour écrire les textes directement dans le corps du programme. Ceci dit, ce n'est pas la meilleure chose à faire. La technique la plus efficace est de taper la description dans un fichier texte. Il suffit ensuite d'une petite routine de chargement pour écrire directement les messages dans le tableau Lieu. Cela ne pose pas de problème si vous connaissez bien les fonctions de gestion de fichier de votre langage. Sinon, jetez un coup d'œil à l'instruction open pour le *Qbasic* et le *Visual Basic*, à la fonction *fopen()* pour le *C* et à la fonction *Create* de *Delphi*.

Compression du texte

Si le jeu est grand, le texte peut consommer une place mémoire non négligeable. Vous pouvez économiser de la place en compressant le texte. Sans utiliser des techniques sophistiquées, vous pouvez remplacer les séquences de caractères les plus fréquentes par des codes numériques plus compactes. Le stockage d'un caractère se fait sous la forme d'un code *ASCII* dont la valeur peut aller de 0 à 255. C'est un gaspillage de place si l'on considère qu'il faut moins de 80 codes pour stocker tous les caractères d'un texte (majuscules, minuscules, minuscules accentuées, signes de ponctuation et chiffres numériques). Il est donc possible de remplacer des combinaisons courantes comme «er» ou «ez» sous la forme d'un code numérique ne prenant qu'un octet (223 par exemple !). En utilisant judicieusement cette technique, vous pouvez compacter de 50% la taille d'un texte. Cela vous oblige à écrire une routine de compression et une routine de décompression, mais le résultat en vaut la peine ! C'est particulièrement le cas en *Qbasic* où la place mémoire est très limitée!

Affichage de la description

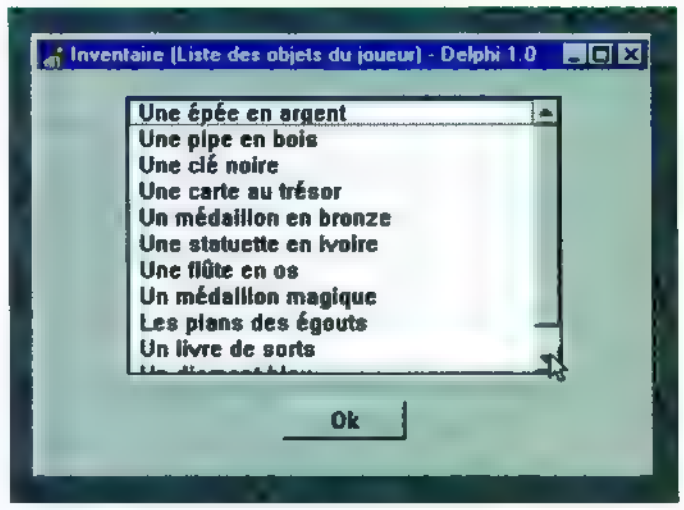
La façon dont l'affichage se fait dépend de la manière dont vous avez conçu votre jeu. Le plus simple est d'utiliser une fenêtre graphique. La gestion d'une fenêtre pose quelques problèmes en *Qbasic* et *Langage C*, mais c'est tout simple à réaliser en *Visual Basic* ou avec *Delphi*. Si vous utilisez le *VB* et que la description risque d'être trop longue, vous pouvez utiliser une fenêtre texte avec ascenseur verticale, comme celle présentée dans l'article sur les trucs et astuces *VB*.

Les liens

Chaque lieu possède un ou plusieurs liens permettant de passer dans d'autres lieux. Par exemple, la place du village possède un lien vers la taverne, un lien vers la forge, un lien la boutique du marchand, un lien vers le poste de garde et un lien vers la sortie du village. La nature physique de ces liens dépend du type de jeu. Par exemple, avec un jeu graphique, chaque lien peut être une zone de l'écran où la souris change de forme. Dans le cas de notre petit logiciel, le plus simple est d'associer un texte à chaque lien et d'afficher ces textes en-dessous de la description du lieu.

Par exemple, la place du village peut avoir la description suivante: «*Vous êtes sur la place d'un charmant petit village. C'est le milieu de l'après-midi, et les gens ne sont pas très actifs. Vous apercevez une taverne, une forge et l'échoppe d'un marchand*». Les liens affichés sous le texte peuvent être «*Visiter la taverne*», «*Entrer dans la forge*», «*Aller voir le marchand*» et «*Quitter le village*». Il suffit de cliquer sur un lien pour passer dans un nouveau lieu.

Un lien est formé par un texte et le numéro du lieu correspondant. Si la taverne est le lieu 12 et la place du village est le lieu 8, le lieu permettant d'aller vers la taverne est le texte «*Visiter la taverne*» et le chiffre 12. De la même manière, il doit y avoir un lien qui permette de quitter la taverne pour retourner sur la place du village («*Quitter la taverne*», suivie de la valeur 8).



▲ *Cet inventaire est écrit en Delphi!*

La programmation des liens est plus compliquée que la description des lieux, car le nombre de liens associés à un lieu est variable. Dans la pratique, cela peut aller de 1 à 5 liens. La technique la plus simple est d'utiliser trois tableaux pour contenir les liens. Un tableau alphanumérique contient le texte des liens, un tableau numérique contient le numéro du lieu contenant le lien, et le second tableau indique l'endroit où mène le lien.

```
' Qbasic
DIM shared TexteLien$(200)
DIM shared EndroitLien(200)
DIM shared NouveauLieu(200)
```

Chaque fois que le programme ajoute un nouveau lien, il le stocke à la suite des liens existants. Cela nécessite que le logiciel conserve dans une variable le nombre de liens existants. Les lecteurs de PC FUTE n°7 reconnaîtront un système de liste. Si le texte d'un lien ne contient qu'une chaîne vide «», cela veut dire que ce lien n'est pas actif. Cela permet de créer un système dynamique où les numéros de liens peuvent évoluer en fonction de l'aventure.

Chaque fois que le programme affiche un lieu, il doit «fouiller» dans la liste des liens pour savoir quels sont les liens correspondants à ce lieu. Par exemple, si le joueur se trouve dans le lieu 45, le programme doit consulter tous les éléments du tableau *Endroit-*

Lien pour déterminer tous les liens correspondants à l'endroit 45. Vous trouvez peut-être que cette technique est plus compliquée qu'une mémorisation simple des liens. C'est exact, mais cela donne un système beaucoup plus souple.

Les liens entre les lieux peuvent évoluer pendant la partie. Par exemple, la découverte d'un passage secret peut amener l'apparition d'un nouveau lien du type «*Entrer dans le passage secret*». Quand cela se produit, le programme n'a qu'à ajouter le lien. Le cas inverse peut également produire. Une avalanche peut boucher l'entrée d'une caverne. Pour gérer cela, le programme peut se contenter d'effacer le lien amenant à la caverne. C'est puissant et assez souple.
Ceci dit, il faut aussi modifier la description du lieu menant à la caverne, pour indiquer qu'il y a un gros tas de rochers sur le chemin.

Création des liens

De la même manière que la description des lieux, les liens peuvent être définis dans un fichier texte ou dans le corps du programme. La seule différence est que les liens n'ont pas de numéros de référence, puisqu'elles sont stockées dans une liste.

Il ne faut pas commettre d'erreurs sur les numéros des lieux, de peur de voir un lien se créer entre une ruelle sordide et la chambre de la princesse ! La création des liens doit se faire avec une routine qui balaye la liste des liens et recherche une place libre, c'est-à-dire un lien dont le texte est vide. Nous avons détaillé le fonctionnement d'une routine de ce type dans le précédent numéro de PC FUTE.

Conversation avec les gens

Un jeu d'aventure n'aurait pas beaucoup d'intérêt s'il n'y avait personne pour discuter. Les systèmes de conversation ne sont généralement pas très sophistiqués. Nous sommes encore loin d'une intelligence artificielle capable de soutenir une vraie conversation ! Notre jeu d'aventure peut utiliser un système de monologue. C'est-à-dire que chaque fois que le joueur indique qu'il veut discuter avec un personnage, celui-ci lui prononce une phrase. Chaque personnage peut posséder un nombre variable de phrases, certaines n'étant que du vent et d'autres contenant des informations utiles.

La technique la plus simple pour gérer cela est d'utiliser encore un système de liste, avec des tableaux jumelés. Chaque phrase est définie par un personnage et un texte. Lorsque le joueur discute avec quelqu'un le programme consulte la liste des phrases, et sélectionne aléatoirement l'une des phrases du personnage.

```
' Qbasic
DIM Shared Personnage(200)
DIM Shared Phrase$(200)
```

Afin d'accélérer les choses, vous pouvez faire une pré-recherche des phrases. Au début d'une conversation, le programme peut «fouiller» dans la liste des phrases et enregistrer dans un tableau le numéro des textes utiles au dialogue. Cette technique permet également de cocher les phrases au fur et à mesure de la conversation, de manière à éviter que le programme ne prononce sans cesse les mêmes paroles.

L'utilisation d'une liste permet d'ajouter ou de retirer facilement des phrases au fur et à mesure de la partie. Par exemple, le joueur trouve une amulette magique. Le programme peut alors rajouter à la liste de phrases du sorcier local «*Tu possède l'amulette du pouvoir. Elle peut vaincre le mal, mais aussi être le mal. Va dans la montagne et rencontre ton destin !*». Et en même temps, le programme peut ajouter un nouveau lien à la montagne, permettant au joueur de pénétrer dans une petite vallée où se trouve les ruines d'une ancienne cité (et un dragon rouge affamé !).

Il est intéressant d'effacer des phrases qui ne servent à plus rien. Par exemple le discours du sorcier sur le médaillon n'est plus utile à partir du moment où le joueur a détruit le dragon (ou s'est fait dévoré !). De la même manière, vous pouvez faire évoluer les formules de salutation au fur et à mesure du déroulement du jeu. Par exemple, le discours du garde peut évoluer de «*Casse-toi, étranger !*» à «*Bien le bonjour, Guerrier !*» et finir par «*Gloire à toi, Pourfendeur du Dragon !*». Du point de vue technique, ce n'est pas grand-chose, mais cela fait plaisir au joueur !

Pour finir

Vous avez maintenant une petite idée sur la manière dont fonctionne un jeu d'aventure. C'est un sujet très complexe, et très intéressant. Il faut créer des lieux, des personnages et des objets. Pensez à tout ce qui peut se produire, imaginez des dialogues amusants, etc.. Si cet article vous inspire et que vous créez quelque chose, n'oubliez pas de nous l'envoyer !

Michel Legrand

Trucs et astuces

LE QBASIC EST LE LANGAGE DES DÉBUTANTS. MAIS IL EXISTE BIEN PEU D'INFORMATIONS SUR CE LANGAGE QUE LES ÉDITEURS JUGENT COMPLÈTEMENT DÉPASSÉ. C'EST POURTANT UN EXCELLENT OUTIL POUR S'INITIER À LA PROGRAMMATION. VOICI LA PRÉSENTATION DE QUELQUES TECHNIQUES INTÉRESSANTES POUR LE PROGRAMMEUR !

Détection de la présence d'un fichier

Le *Qbasic* génère une erreur d'exécution si vous essayez d'ouvrir en lecture un fichier qui n'existe pas. Il n'existe pas d'instruction permettant de vérifier si un fichier existe vraiment sur le disque. Mais il est possible d'écrire une routine qui lui supplée, en utilisant le système de gestion des erreurs.

En cas d'erreur, le *Qbasic* stoppe le déroulement du programme et affiche un message d'erreur approprié. L'instruction *ON ERROR GOTO* permet de modifier, en indiquant au *Qbasic* un label sur lequel il doit se brancher en cas d'erreur. Cela permet d'écrire un module de traitement des erreurs dans le programme même. Une fois le traitement terminé, l'instruction *RESUME NEXT* indique au *Qbasic* de reprendre le cours normal du programme.

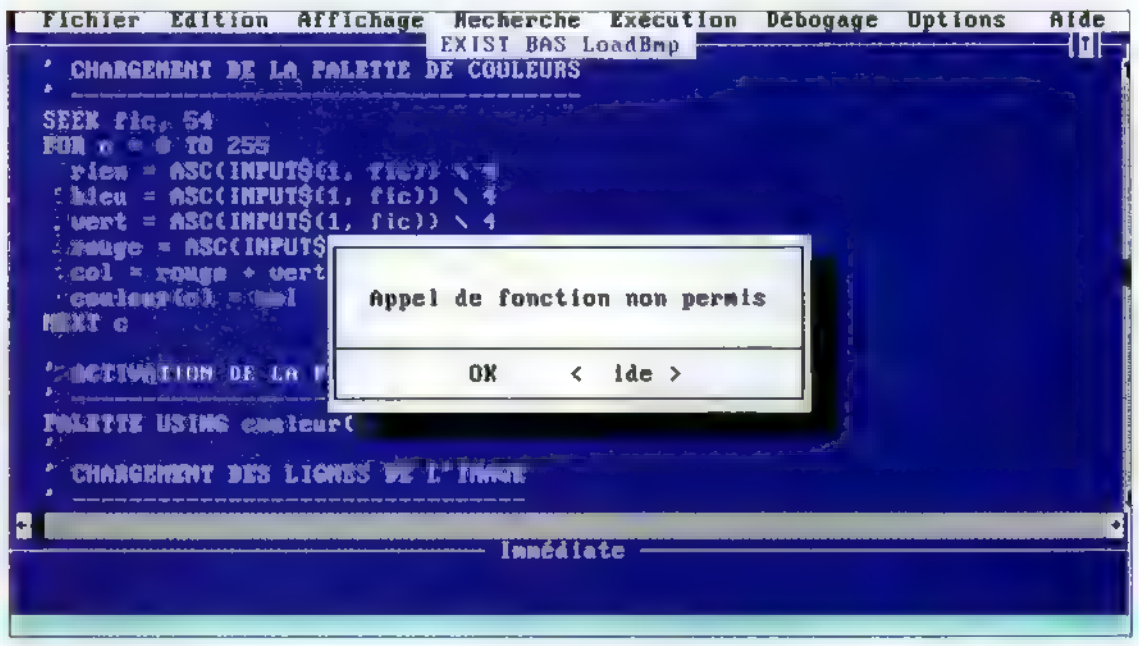
```
ON ERROR GOTO GestionErreur:

GestionErreur:
  .....
  RESUME NEXT
```

La fonction *TestFichier* utilise cette technique pour vérifier si un fichier existe vraiment sur disque. Elle commence par écrire la valeur 1 dans la variable globale *DetectionFichier*, puis elle ouvre le fichier en lecture et le referme. Le programme retourne ensuite le contenu de la variable *DetectionFichier*. Si une erreur d'ouverture se produit, le programme exécute les instructions définies par le label *GestionErreur*. Le programme de traitement d'erreur est très limité et se contente d'écrire la valeur 0 dans la variable globale *DetectionFichier*.

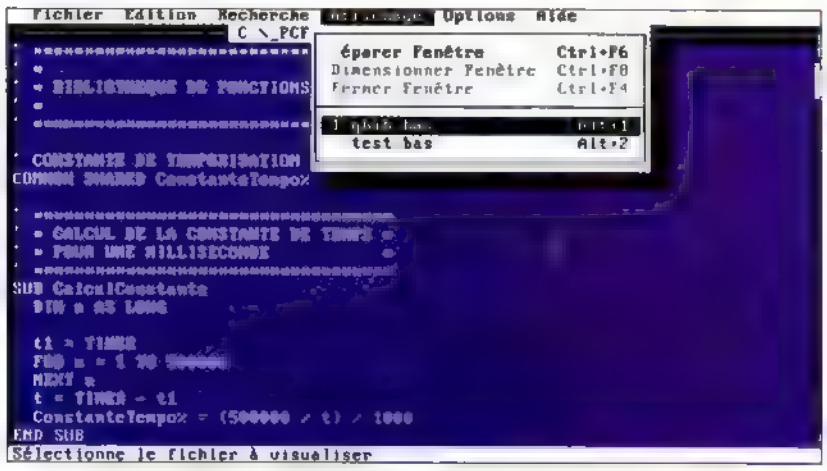
```
FUNCTION TestFichier (nom$)
 DIM rep AS INTEGER
 DIM fic AS INTEGER
 ON ERROR GOTO GestionErreur:
 DetectionFichier = 1
 fic = FREEFILE
 OPEN nom$ FOR INPUT AS fic
 IF DetectionFichier = 1 THEN
   CLOSE fic
 END IF
 Testfichier = DetectionFichier
END FUNCTION

GestionErreur:
  DetectionFichier = 0
  RESUME NEXT
```

Notre fonction retourne donc la valeur 1 si le fichier testé est présent sur la disquette et 0 dans le cas contraire. Voici un petit programme d'exemple qui vous montre comment utiliser la fonction *TestFichier* dans vos programmes :

```
' ******************************
' * UTILISATION DE TestFichier *
' ******************************
DIM SHARED DetectionFichier

CLS
```



▲ *Le Qbasic produit ce genre d'erreur quand un programme tente d'accéder au contenu d'un fichier inexistant.*



▲ *L'éditeur EDIT permet de charger plusieurs fichiers en mémoire. Pour passer d'un listing à un autre, il faut utiliser la souris et cliquer sur le nom du second listing. Cette image vous montre un exemple où le fichier en cours d'édition est qlib.bas. Pour éditer le fichier test.bas, il suffit de cliquer sur son nom.*

```
nom$ = «test.txt»

IF Testfichier(nom$) = 1 THEN
  PRINT «Fichier présent!»
ELSE
 PRINT «Fichier inexistant!»
END IF
END
```

Nous vous conseillons de tester systématiquement la présence des fichiers avant de les charger. C'est une mesure de sécurité indispensable pour éviter les problèmes. Vous trouverez sur le disque un programme d'exemple dont la tâche est d'afficher sur l'écran le contenu d'un fichier texte. Il affiche un message d'alarme si le fichier n'existe pas.

Création d'une bibliothèque de routines

La plupart des programmes ont des points communs, ne serait-ce qu'au niveau de l'affichage, de la gestion du clavier et des fichiers. C'est pourquoi le programmeur astucieux peut gagner du temps en écrivant des routines standard et en les réutilisant dans tous ses programmes. C'est par exemple le cas pour les routines graphiques, la gestion de la souris, les routines de temporisation, etc.

L'éditeur du *Qbasic* n'est pas prévu pour faciliter la récupération des routines d'un programme vers un autre programme. Il est possible de le faire à l'aide des fonctions *Copier/Coller*, mais ce n'est pas très pratique. Il faut charger le premier programme, sélectionner la routine à copier avec la souris, cliquer sur la fonction *Copier*, charger le second programme et utiliser la fonction *Coller* pour intégrer la routine dans le nouveau listing. L'opération doit être répétée pour chaque routine. Les concepteurs du *Qbasic* auraient pu avoir un peu d'imagination et inventer quelque chose de plus pratique !

Heureusement, on peut parvenir au même résultat, grâce à une petite astuce. Les fichiers programmes du *Qbasic* sont sauvés sous forme de fichiers textes. Il est possible d'utiliser l'éditeur de textes du *DOS* pour les modifier. Pour lancer cet éditeur, il suffit de taper sous *DOS* la commande suivante :

```
EDIT
```

L'éditeur *texte* affiche la totalité du programme sur l'écran. C'est plus facile à utiliser que l'éditeur du *Qbasic* qui n'affiche qu'une routine à la fois. Mais le plus intéressant est la possibilité de charger plusieurs fichiers en même temps dans l'éditeur. Pour transférer des routines entre deux programmes, il suffit de charger les deux listings en mémoire et de procéder à des opérations de *Copier/Coller*. Grâce à l'affichage du listing en pleine page, il est possible de sélectionner plusieurs routines en même temps et de les transférer en une seule opération de *Copier/Coller*.

Ce n'est pas très pratique que de devoir fouiller les listings *BAS* à la recherche d'une routine particulière. Il est plus rationnel de copier toutes les routines intéressantes dans un seul fichier, de manière à constituer une bibliothèque de routines complète. Cela facilite la récupération des routines et accélère le travail de programmation. Vous trouverez sur le disque une bibliothèque portant le nom de *Qlib.bas*. Elle contient diverses routines de bases, indispensables pour écrire des programmes en *Qbasic*. Vous pouvez récupérer ces routines et les copier dans vos propres programmes avec l'éditeur *EDIT*.

Les bibliothèques ne contiennent pas nécessairement que des routines. Elles peuvent aussi contenir la déclaration de variables globales et même de tableaux complets. Par exemple, on peut imaginer qu'une bibliothèque ne contienne qu'un tableau et trois routines permettant de lire et d'écrire des informations dans le tableau. Dans le même ordre d'esprit, une bibliothèque peut également contenir des constantes de tout type.
Notre bibliothèque est loin d'être complète. Elle ne contient qu'une quinzaine de routines plus ou moins utiles. Vous pouvez l'améliorer avec vos propres routines de base. Il est possible de créer plusieurs bibliothèques si le nombre de fonctions devient trop important. Par exemple, il peut y avoir une bibliothèque pour le graphisme, une bibliothèque pour la gestion de fichiers, une bibliothèque pour la gestion du clavier, etc.

Mike Terranova

Vous avez des trucs et astuces en Qbasic ? Vous avez écrit des routines intéressantes ? Vous êtes l'auteur d'un programme très utile ? Envoyez-nous vos créations pour que tout le monde en profite ! Envoyez-nous également vos questions. Nous avons besoin de connaître les problèmes que vous rencontrez en Qbasic pour vous aider.

DELPHI EST-IL UN VÉRITABLE LANGAGE DE PROGRAMMATION OU UN GÉNÉRATEUR DE BASE DE DONNÉES ? EST-CE UN VÉRITABLE CONCURRENT POUR LE VISUAL BASIC ? POUR CONNAÎTRE LA RÉPONSE À CES QUESTIONS, NOUS AVONS MIS UN EXEMPLAIRE DE DELPHI ENTRE LES MAINS D'UN SPÉCIALISTE EN VISUAL BASIC. LE RÉSULTAT DU TEST EST ÉTONNANT !

L'épreuve du feu

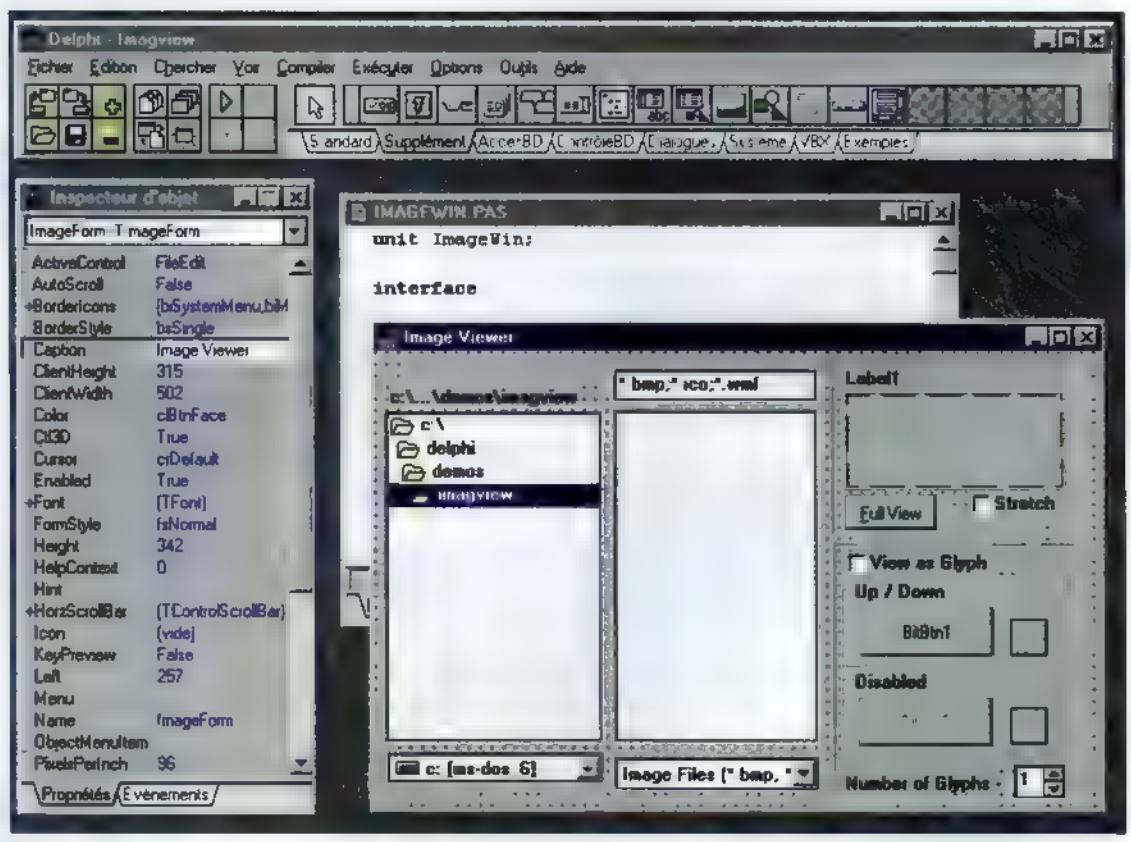
Windows est un système très intéressant, mais incroyablement difficile à programmer. C'est pourquoi j'ai été particulièrement heureux de découvrir le *Visual Basic*, voici deux ans. Enfin un langage puissant permettant d'écrire facilement des logiciels *Windows* ! Je me suis amusé à créer toutes sortes de programmes amusants. Pendant ce temps, *Borland* sortait la première version de *Delphi*, en la présentant comme un système permettant de créer des bases de données. N'étant pas intéressé par le sujet, j'ai vaguement regardé les publicités, en me disant que *Borland* avait bien périclité depuis le temps du *Turbo Pascal* et du premier *Turbo C*. C'est pourquoi j'ai haussé les épaules quand la Rédaction de PC FUTE a posé un exemplaire de *Delphi* sur mon bureau et m'a demandé d'écrire un papier sur mes impressions. Mes sujets de prédilections sont le graphisme, les jeux et l'intelligence artificielle et non la gestion !

Installation

Le CD-ROM d'installation qui m'a été fourni contient *Delphi 2.0* version client-serveur. C'est le nec plus ultra des versions de *Delphi* et il nécessite *Windows 95*. Heureusement, le même CD-ROM contient aussi *Delphi 1.0* pour *Windows 3.1*. C'est cette version que j'ai installée, puisque de nombreux lecteurs utilisent encore *Windows 3.1*. Ceci dit, le test a été effectué avec *Windows 95*, puisque c'est le seul système qui tourne sur ma machine depuis plus d'un an. L'installation complète s'est faite sans problème, et occupe environ 80 Mo de disque dur.

Le choc de la couverture

Il ne m'a fallu que quelques minutes pour être séduit par *Delphi*. Ce n'est pas un système de gestion de base de données, mais un langage de programmation très performant, permettant d'écrire n'importe quel type de programme *Windows* ! C'est un mélange entre le *Visual Basic* et un excellent *Pascal*. Je ne comprends pas pourquoi les gens de *Borland* ont appelé ce système *Delphi*, à la place de «*Visual Turbo Pascal*» ? C'est probablement une question de position-



▲ *La programmation en Delphi consiste à poser des objets graphiques sur l'écran et à définir le code qui s'exécute lorsque l'utilisateur clique sur ces objets.*

nement commercial par rapport au *Visual Basic*. Mais cela a pour effet de laisser dans l'ignorance des clients potentiels comme moi ! Il a fallu un coup de chance et l'aide de PC FUTE pour que je comprenne que le langage de mes rêves existait et que ce n'était pas le *Visual Basic* !

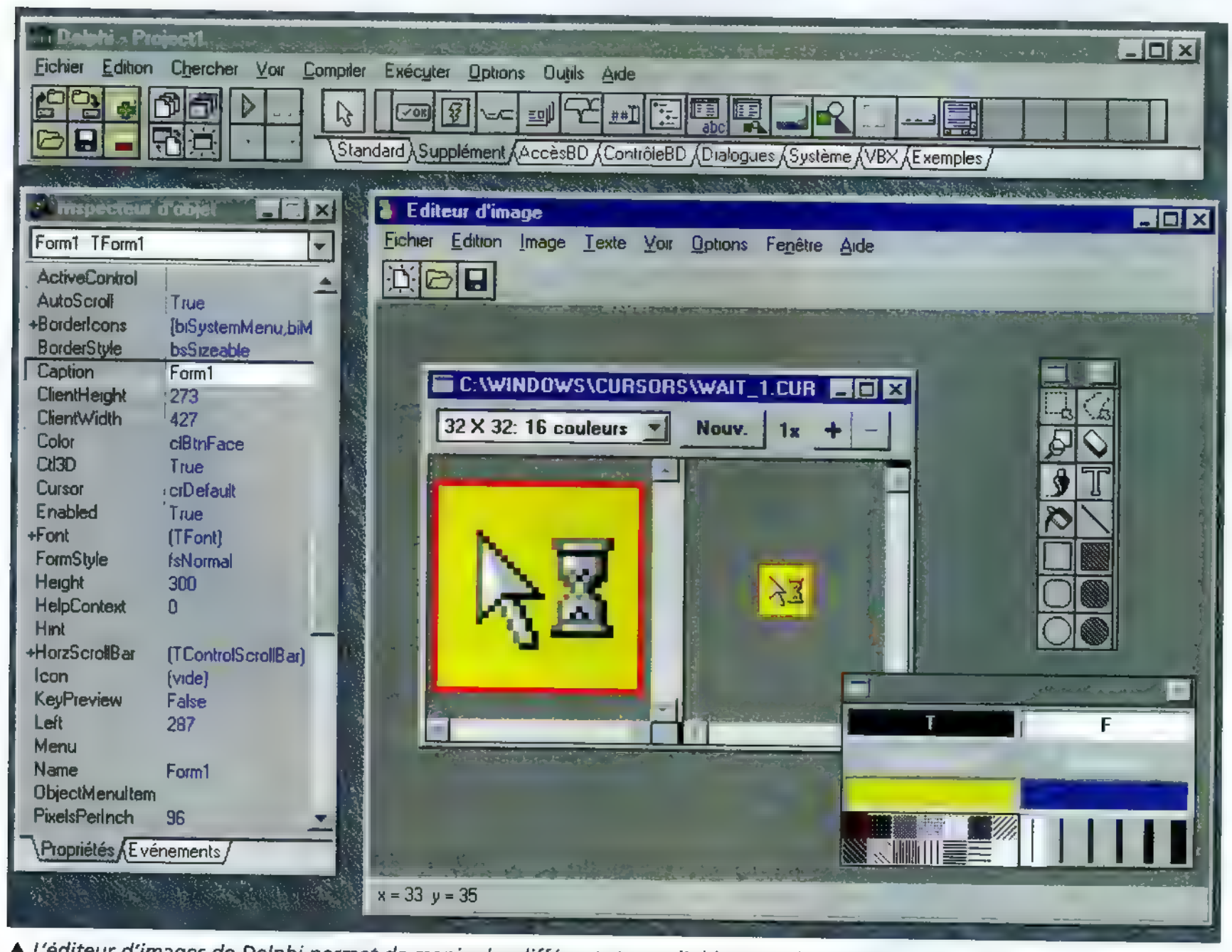
Le système visuel

La philosophie d'utilisation de *Delphi* est identique à celle du *Visual Basic*. La programmation se fait en posant des objets graphiques sur une feuille et en écrivant le code correspondant aux actions possibles. C'est l'enfance de l'art et il suffit de quelques minutes pour écrire un premier programme, aussi bien en *Visual Basic* qu'avec *Delphi*.
Les composants graphiques de *Delphi* sont nombreux et bien conçus. Il y a des boutons, des zones de textes, des zones graphiques, des timers, etc. Ils ont le gros avantage d'être en relief, ce qui n'est pas le cas avec la version de base du *VB 3.0* et n'apparaît qu'avec le *VB 3.0 Pro* et le *VB 4.0*.
Il y a également tout plein de choses intéressantes, comme les boutons *Bitmaps* qui peuvent intégrer des images sur les boutons eux-mêmes. Encore plus fort ! Il est possible d'utiliser les contrôles *VBX* du *Visual Basic* avec *Delphi* !

La programmation

La différence entre les deux systèmes se fait sentir au niveau du langage. Le langage sous-jacent du *Visual Basic* est un basic amélioré, mais un basic quand même ! En revanche, le langage interne de *Delphi* est une version améliorée du *Turbo Pascal*, l'un des meilleurs langages de programmation existants. Sinon, le principe est exactement le même. Le programmeur sélectionne un objet graphique et tape le code correspondant à l'action sur l'objet.

L'éditeur de *Delphi* présente l'avantage d'éditer le programme dans une



▲ *L'éditeur d'images de Delphi permet de manipuler différents types d'objets graphiques. Par exemple, il sert ici à créer un nouveau curseur de souris.*

seule fenêtre. Il suffit de cliquer sur un objet pour que l'éditeur affiche automatiquement les lignes de codes correspondantes. C'est beaucoup plus pratique que le *Visual Basic* où l'éditeur ne peut afficher qu'une routine à la fois, et où il faut passer par un système compliqué de listes de fonctions pour obtenir le code d'une routine particulière. En *Delphi*, il est possible de mélanger librement des «routines d'interfaces», liées à un composant graphique et des routines classiques.

Un autre avantage de *Delphi* est la vérification automatique des variables. Il arrive fréquemment que l'on se trompe dans le nom d'une variable. Par exemple, *nb_sprites* peut être écrite *nb_sprite*. Les variables de *Delphi* doivent obligatoirement être déclarées avant leur utilisation, ce qui permet au langage de contrôler automatiquement la validité des variables du programme et de signaler les éventuelles erreurs. Cela manque beaucoup au *Visual Basic*, où les «variables fantômes» provenant d'une erreur de frappe contiennent toujours la valeur 0, ce qui provoque des erreurs ! Il est possible de remédier à cela, en utilisant l'option *Explicit* qui force le *VB* à contrôler les variables.

La programmation objet

Delphi est un langage objet, c'est-à-dire qu'il est possible de créer des objets informatiques à partir d'autres objets. C'est une technique de programmation moderne qui a le vent en poupe. Cependant, il est possible d'écrire des programmes en *Delphi*, sans passer par les subtilités de la programmation objet. Une connaissance minimale du *Pascal* suffit pour écrire des logiciels performants. Cela est intéressant quand on considère le nombre de personnes qui ont travaillé avec le *Turbo Pascal* pour écrire des programmes *DOS*.

La programmation objet permet de réaliser des choses très intéressantes. Par exemple, il est possible de créer de nouveaux composants graphiques à partir des composants de base. D'ailleurs, tous les composants de base (boutons, fenêtres graphiques, fenêtres textes, sélecteurs de fichiers, boîtes de dialogue, etc.) ont été créés en *Delphi*, à partir de primitives de base. Les listings sont fournis sur le CD-ROM. C'est très intéressant pour comprendre comment cela fonctionne et créer ses propres composants.

Environnement de développement

L'environnement de développement est une belle mécanique bien huilée. Les gens de *Borland* ont dû travailler pendant très longtemps pour arriver à ce degré de simplicité et de puissance. Il y a quelques outils intéressants, comme un éditeur d'images pouvant manipuler aussi bien des images *BMP*, des icônes et des curseurs souris que des fichiers de ressources.

Le compilateur de *Delphi* génère un véritable programme exécutable. C'est plus pratique et plus rapide à l'exécution que le système de code intermédiaire produit par le *VB* qui nécessite un runtime pour fonctionner. En revanche, les programmes sont plus volumineux, ce qui est normal, dans la mesure où chaque logiciel contient toutes les routines dont il a besoin pour fonctionner.

Conversion VB vers Delphi

Puisque la philosophie des deux langages est très proche et que *Delphi* peut être considéré comme une sorte de sur-ensemble du *VB*, il est facile de convertir des programmes *VB* en *Delphi*. Il suffit de faire la correspondance entre les contrôles du *VB* et les composants *Delphi* et d'adapter le code. *Delphi 1.0* est livré avec le fichier *Vb2delph.wri* de 112 Ko qui explique tout ce qu'il faut savoir pour adapter des programmes *VB* en *Delphi*. En revanche, il est très difficile d'adapter un programme *Delphi* en *VB*, surtout un source écrit en programmation objet.

Pour finir

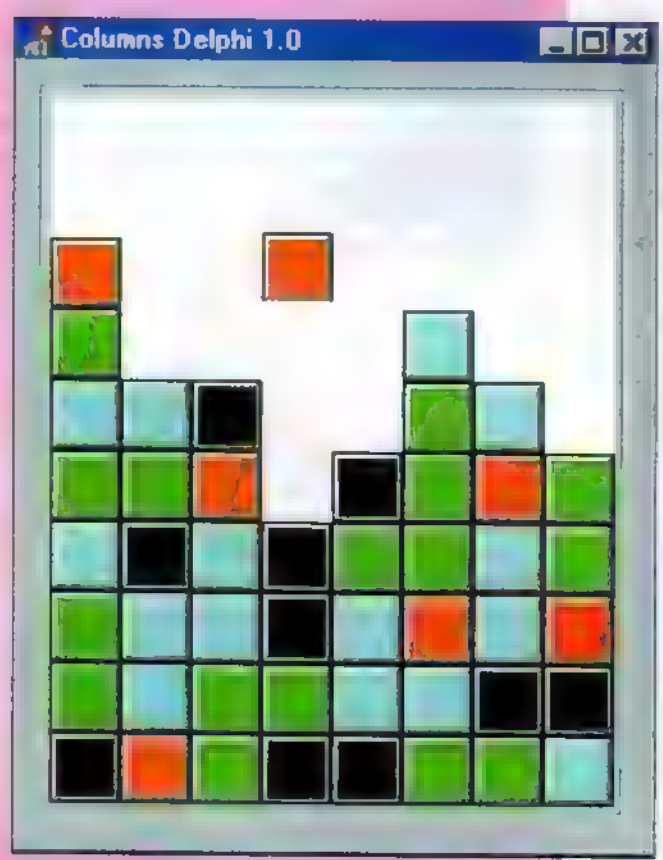
Pour moi, *Delphi* est le langage du siècle ! Il est capable de faire tout ce que fait le *Visual Basic* et d'aller bien au-delà ! Il permet d'exploiter toute la puissance de *Windows* et de faire des choses formidables. Depuis que j'ai découvert ce langage, j'ai jeté mon *Visual Basic* et je ne fais plus que du *Delphi*.

Marc Lenoir

Delphi, créer votre propre Colums

La puissance de Delphi rend aisée la réalisation de petits programmes graphiques. Nous avons pris comme exemple le jeu Columns, très connu et très facile à réaliser. Ce logiciel convient parfaitement pour illustrer les fonctions graphiques de Delphi.

Dès sa sortie, le jeu russe *Tétris* a connu un immense succès. De nombreux programmeurs ont repris le thème du jeu pour créer d'autres logiciels. L'un des plus simples et des mieux réussis est *Columns*. Le joueur se retrouve dans un espace de jeu où des cubes tombent du haut de l'écran. Le but du jeu est d'orienter la chute de manière à créer des alignements de 3 cubes de même couleur, provoquant ainsi leur disparition ! Le jeu s'arrête quand le tas de cubes touche le haut de l'écran. Il existe plusieurs variantes de *Columns*. Les différences se font sur la taille des cubes, le nombre de cubes qui tombent en même temps et le nombre de cubes à disposer pour réaliser un alignement.



▲ *Columns est un jeu simple où le but est d'orienter la chute des cubes, de manière à former des alignements de cubes.*

Notre Columns

En regardant les images accompagnant cet article, vous pouvez voir à quoi ressemble notre *Columns*. Les cubes tombent du haut de l'écran et s'entassent en bas. Lorsqu'un cube apparaît en bas de l'écran, il se met à tomber à la vitesse d'une case par seconde. La chute s'arrête lorsque le cube touche le bas de l'écran ou est bloqué par un autre cube. Pendant la chute, il est possible de déplacer le cube vers la droite ou vers la gauche, grâce aux flèches du clavier. De la même manière, il est possible d'accélérer la chute en pressant sur la barre d'espace ou sur la flèche basse. Cela évite au joueur d'avoir à attendre la fin de la chute alors que le cube est placé sur la bonne colonne. Chaque fois que le programme détecte un alignement de 3 cubes de même couleur, il les fait disparaître.

Quelques problèmes techniques

Les règles du jeu sont simples. Mais la programmation pose un certain nombre de problèmes techniques. Les principaux sont : la gestion du mode graphique, la lecture du clavier, le dessin des cubes, le dessin de la fenêtre de jeu en creux, le déplacement des cubes sur l'écran, la détection et l'effacement des alignements de cubes identiques. Tout cela n'est pas simple à programmer. Nous allons étudier chaque cas individuellement et voir comment il peut être résolu.

Affichage des cubes

Il existe différentes techniques pour dessiner les cubes sur l'écran. Par exemple, chaque cube peut être un composant de type *Shape* (forme géométrique élémentaire) ou une image de type *Timage*. Il est possible de disposer une matrice de composants sur l'écran et de n'activer que les composants correspondants à un cube, grâce à la propriété *Visible*. Néanmoins, ce n'est pas la technique choisie pour le logiciel.

La plupart des composants graphiques de *Delphi* possèdent la propriété *Canvas*. Celle-ci permet au programme d'accéder au contenu de l'image bitmap du composant. Il est possible de faire toutes sortes d'opé-

rations graphiques sur un *Canvas* (tracé de cercles, dessin de rectangles, affichage de textes, etc.). En tout, il existe 17 méthodes différentes pour modifier le contenu d'un *Canvas*. La méthode que nous allons utiliser pour dessiner les cubes est *Draw*. Cette méthode permet de copier un élément graphique sur un *Canvas*. L'élément graphique peut être une image ou une icône. La syntaxe suivante vous montre comment afficher une icône à une position quelconque d'un objet graphique :

```
Objet. Canvas.Draw(px, py,
                   Icone);
```

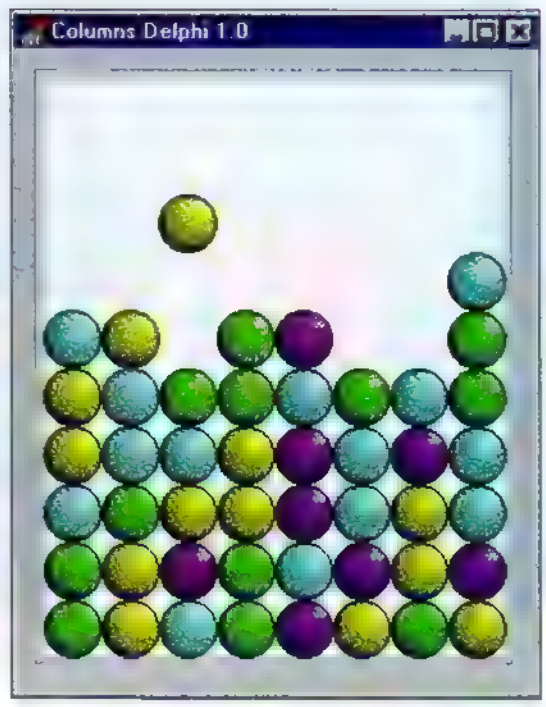
Par exemple, vous pouvez utiliser cette technique pour dessiner une icône directement sur une fiche. La syntaxe suivante vous montre comment dessiner l'icône *Icone* sur la fiche *Form1.* :

```
Form1.Canvas.Draw(px, py,
                  Icone);
```

L'objet Icone doit être un objet de type Ticon. Le programme suivant vous montre comment créer et utiliser une variable de ce type :

```
var Icone : Ticon;
    Nom   : String;
      Nom := 'Icone1.ico';
Icone := TIcon.create;
Icone.LoadFromFile(nom);
```

Le fait de déclarer une variable de type *Ticon* ne réserve pas la mémoire nécessaire au stockage de l'icône. Pour cela, il faut utiliser la méthode *Ticon.create*. La méthode *LoadFromFile* permet de charger une icône dans une variable de type *Ticon*. Il suffit ensuite d'utiliser la méthode *Draw* pour dessiner l'icône à un emplacement quelconque de la fiche.



▲ *Vous pouvez utiliser d'autres icônes afin de donner un look différent au jeu.*

```
var Icone : Ticon;
    px, py: Integer;
      (* Chargement icône *)
...
px := 20;
py := 20;
Form1.Canvas.Draw(px, py,
                  Icone);
```

Cette technique fonctionne aussi avec des images *BMP*, à condition d'utiliser des objets *Timage* et non des objets *Ticon*. Il est plus pratique d'utiliser des icônes pour notre jeu car la bibliothèque d'icônes pour *Windows* est presque illimitée !

Les cubes du logiciel sont des icônes 32x32 pixels. Nous les avons fabriquées à l'aide de l'outil de dessin de *Delphi*. Elles sont stockées en mémoire dans le tableau *Icone[]* dont voici la définition :

```
const MaxIcones = 4;
var Icone: array[0..MaxIcones]
           of Ticon;
```

Le chargement d'une icône en mémoire est un peu compliqué car il faut fournir le chemin complet du fichier (par exemple, *'C:\PCFUTE\COLUMNS\ICONE1.ICO*). C'est toujours une mauvaise idée que d'utiliser un chemin en dur dans un programme. Cela peut provoquer une catastrophe en cas de changement de répertoire. Il est plus astucieux de stocker les fichiers dans le même répertoire que le programme et de récupérer le chemin du logiciel au moment du chargement. C'est la technique utilisée dans la routine suivante :

```
procedure Charger_icone(
          n: integer;
          nom:string);
var chemin : string;
    fic    : string;
begin
 GetDir(0, chemin);
 Icone[n] := TIcon.create;
 fic := chemin + '\' + nom;
 Icone[n].LoadFromFile(fic);
end;
```

Cette routine réserve automatiquement la place nécessaire à une icône et la charge dans le tableau *Icone[]*. Elle est utilisée parle programme pour stocker toutes les icônes du programme. Il y a 4cubes de couleur et une cinquième icône qui contient une zone blanche de 32x32 pixels. Cette image correspond à l'absence de cube, ou plutôt du cube vide. Elle permet d'effacer l'image d'un cube sur l'écran. Ce «non-cube» est stocké à la position 0 du tableau *Icone[]*.

```
charger_icone(0, 'vide.ico');
charger_icone(1, 'bleu.ico');
charger_icone(2, 'vert.ico');
charger_icone(3, 'rouge.ico');
charger_icone(4, 'noir.ico');
```

L'écran de jeu est organisé en 10 lignes de 8 colonnes. Cela représente 80 positions différentes d'affichage pour les cubes. Ces valeurs sont stockées dans les constantes *MaxLignes* et *MaxColonnes*.

```
const
  MaxLignes = 10;
  MaxColonnes = 8;
```

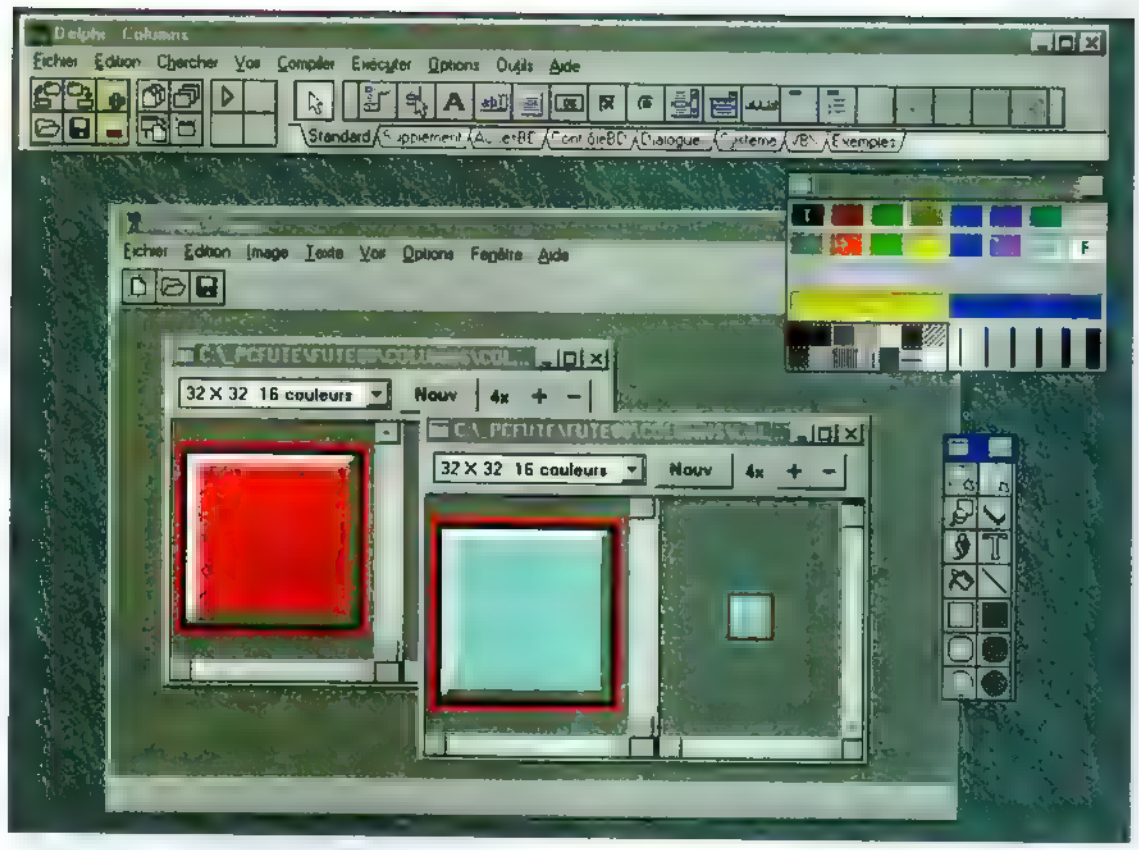
La routine *aff_element* permet de dessiner un cube sur l'écran. Les paramètres sont la position (ligne, colonne) du cube et sa couleur. Il s'agit en fait de son numéro de référence qui doit être compris entre 1 et 4, puisqu'il n'y a que 4 icônes en mémoire (plus l'icône vide !).

```
procedure aff_element(
          ligne, colonne,
          couleur :integer);
var px: integer;
    py: integer;
begin
 px := X0_JEU+(colonne-1)*32;
 py := Y0_JEU+(ligne-1)*32;
 Form1.Canvas.Draw(px, py,
                   Icone[couleur]);
end;
```

Les icônes ont une taille de 32x32 pixels. C'est pourquoi le programme multiplie la ligne et la colonne du cube par 32 pour obtenir la position d'affichage en pixel. La numérotation des lignes et des colonnes commence à la valeur 1, ce qui pose un petit problème de calcul. En effet, le calcul donne la position en pixels (32, 32) pour le cube situé à la ligne 1 et à la colonne 1.
C'est pour éviter cela que le programme diminue systématiquement de 1 les numéros de ligne et de colonne.

Les constantes *X0_JEU* et *Y0_JEU* définissent la position d'affichage du premier cube. En jouant sur ces deux valeurs, il est possible de placer l'espace de jeu à n'importe quel emplacement de l'écran.

```
const
  X0_JEU = 16;
  Y0_JEU = 16;
```



▲ *Les icônes des cubes ont été dessinées avec l'éditeur d'images de Delphi.*

Déplacement d'un cube

Nous savons maintenant comment afficher un cube sur l'écran à une position quelconque. A partir de cela, il est facile d'écrire une routine qui déplace le cube vers le bas de l'écran. Avant toute chose, il faut un mécanisme qui puisse stocker la position et le type du cube. Pour cela, il faut utiliser un ensemble de 3variables globales :

```
var  ligne_cube:   integer;
  colonne_cube: integer;
  cube_courant: integer;
```

Pour afficher un cube sur l'écran, il faut définir ces 3 variables et appeler la routine *aff_cube*. Elle utilise la routine *aff_element* pour dessiner l'icône correspondante sur l'écran.

```
procedure aff_cube;
begin
 aff_element(ligne_cube,
             colonne_cube,
             cube_courant);
end;
```

La routine *eff_cube* efface automatiquement le cube courant. En fait, elle affiche le cube vide (de numéro 0) à la position courante.

```
procedure eff_cube;
begin
 aff_element(ligne_cube,
             colonne_cube,
             0);
end;
```

La routine *nouveau_cube* permet de générer aléatoirement un nouveau cube. Pour cela, elle détermine aléatoirement le type de cube et la colonne où il apparaît. La variable *ligne_courante* est toujours fixée à 1 car le cube doit nécessairement apparaître en haut de l'écran. Le tirage aléatoire se fait avec l'instruction *random(n)*qui génère un nombre aléatoire compris entre 0 et n-1. Par exemple, *random(4)* fournit un nombre aléatoire compris entre 0 et 3. Cela ne convient pas à notre logiciel, puisque les icônes et les colonnes sont numérotées à partir de la valeur 1. C'est pourquoi le logiciel ajoute systématiquement 1 à chaque tirage. La formule *random(4)+1* donne un nombre aléatoire compris entre 1 et 4, ce qui correspond au but recherché !

```
procedure nouveau_cube;
begin
 cube_courant :=
    random(MaxIcones)+1;
 ligne_cube := 1;
 colonne_cube :=    random(MaxCo-
```

```
lonnes)+1;
 aff_cube;
end;
```

Pour faire chuter le cube courant d'une case vers le bas, il faut l'effacer, augmenter de 1 la valeur de la variable *ligne_courante* et afficher la nouvelle position du cube. Cela se fait de la manière suivante :

```
eff_cube;
ligne_cube := ligne_cube +1;
aff_cube;
```

Comme vous pouvez le voir, c'est très facile à réaliser !Néanmoins, la chute ne peut continuer éternellement et doit s'arrêter en bas de l'écran. L'arrêt se produit lorsque le cube se trouve sur la dernière ligne de l'écran, c'est-à-dire lorsque le contenu de la variable *ligne_cube* est égal à la constante *MaxLignes*.

```
if ligne_cube<>MaxLignes then
begin
 eff_cube;
 ligne_cube := ligne_cube +1;
 aff_cube;
end;
```

Réglage de la vitesse de chute

Les cubes doivent tomber les uns après les autres, d'une manière régulière. La seule manière de réaliser cela est d'utiliser un timer. C'est un composant *Delphi* qui exécute une routine à intervalle régulier. Le délai entre deux exécutions peut varier de quelques millisecondes à plus d'une minute.

La routine *TForm1.Timer1Timer* est associée au composant *Timer1*dont l'intervalle de temps est de 300 ms, ce qui correspond à une activation toutes les 0,3 secondes. Sa tâche est très simple, puisqu'elle se contente d'appeler la routine *aller_bas* qui gère le déplacement du cube d'une ligne vers le bas. C'est ce que l'on appelle une routine d'interface. Elle assure la liaison entre le corps du programme et un composant graphique.

```
procedure TForm1.Timer1Timer(
            Sender: TObject);
begin
 aller_bas;
end;
```

La vitesse de chute est fonction du délai d'activation du composant *Timer1*. Plus le temps est court, plus les cubes tombent vite. Vous pouvez utiliser cet effet pour augmenter la vitesse de jeu en cours de partie. Il suffit pour cela de gérer un score et de diminuer la vitesse chaque fois que le score passe une certaine limite.

```
(* VITESSE INITIALE *)
Timer1.Interval := 300;
....
(* AUGMENTATION VITESSE *)
Timer1.Interval := 200;
....
(* VITESSE DE LA MORT *)
Timer1.Interval := 50;
```

Il est très important d'utiliser un composant timer pour régler la chute des cubes. Cela assure une vitesse de jeu constante sur tous les types de PC. Les différences de performances sont considérables entre un 486 et un Pentium, ainsi qu'entre une carte graphique classique et une carte accélératrice haut de gamme. Les cubes ne s'afficheront pas à la même vitesse sur tous les PC, mais le jeu aura une vitesse de réaction constante.

La routine *aller_bas* descend le cube d'une ligne vers le bas. Elle utilise la fonction *test_bas* pour savoir si la chute est possible. Il est possible qu'un autre cube bloque le passage ou que le cube se trouve sur la dernière ligne de l'écran. Si c'est le cas, elle appelle la procédure *fin_chute* pour signaler au programme que le cube est bloqué et utilise la fonction *exit* pour provoquer la sortie immédiat de la routine.

```
procedure aller_bas;
begin
 if test_bas=false then
 begin
   fin_chute;
   exit;
 end;

 (* CHUTE DE 1 LIGNE *)
 eff_cube;
 ligne_cube := ligne_cube+1;
 aff_cube
end;
```

La fonction *test_bas* retourne la valeur *false* si le cube se trouve sur la dernière ligne de l'écran ou si un autre cube bloque la chute. Elle retourne la valeur *true* si le déplacement peut se faire sans contrainte.

```
function test_bas:boolean;
var ligne  :integer;
    colonne:integer;
begin
 (* TEST SI DERNIERE LIGNE *)
 if ligne_cube = MaxLignes then
 begin
   test_bas:=false;
   exit;
 end;
 (* TEST SI CUBE EN BAS *)
 ligne:=ligne_cube+1;
 colonne:=colonne_cube;
 if jeu[ligne,colonne]<>0 then
 begin
   test_bas := false;
   exit;
 end;
 test_bas:=true;
end;
```

Pour que cette routine fonctionne, le programme doit garder en mémoire la position des cubes bloqués. Le plus

suite p. 55

Manipulations graphiques en VB

LE VB EST ASSEZ LIMITÉ AU POINT DE VUE MANIPULATIONS D'IMAGES. IL PERMET JUSTE DE COPIER UNE IMAGE D'UN CONTRÔLE GRAPHIQUE À UN AUTRE. HEUREUSEMENT, IL EST POSSIBLE D'UTILISER UNE FONCTION GRAPHIQUE DE WINDOWS POUR ÉTENDRE LES POSSIBILITÉS DU VB.

Nous avons vu dans le dernier numéro de PC FUTE que l'une des nouveautés du *Visual Basic 4.0* est la fonction *PaintPicture*. Elle permet de copier des blocs de pixels d'un contrôle graphique sur un autre, ce qui autorise la réalisation d'effets graphiques très intéressants. Il est possible de faire la même chose en *Visual Basic 3.0*, en utilisant la fonction graphique *BilBlt*.

Cette fonction fait partie de la bibliothèque graphique de base de *Windows*. Elle est définie dans le fichier *GDI.DLL*. Pour l'utiliser dans un programme en *VB*, il faut créer un module *BAS* et taper la déclaration suivante :

```
Declare Function BitBlt Lib «gdi»
        (ByVal hDestDC%,
         ByVal X%, ByVal Y%,
         ByVal nWidth%,
         ByVal nHeight%,
         ByVal hSrcDC%,
         ByVal XSrc%, ByVal YSrc%,
         ByVal dwRop&) As Integer
```

Cela indique au *VB* que la fonction *BitBlt* se trouve dans le fichier *DLL* portant le nom de *GDI*. Cette fonction permet de copier un bloc de pixels d'une image vers une autre. La syntaxe est conforme à celle donnée par Microsoft dans sa documentation technique. Voici sa signification :

```
hDestDC%: Handle de l'image de destination
X% et Y%: Position d'affichage du bloc de pixels
nWidth%: Largeur du bloc de pixels
nHeight%: Hauteur du bloc de pixels
hSrcDC%: Handle de l'image d'origine
XSrc% et Ysrc%: Position d'origine du bloc
dwRop&: Opérateur logique de transfert
```

Les images manipulées par le *VB* sont stockés dans des contrôles graphiques. Mais *Windows* ne sait pas ce qu'est un contrôle graphique ! Pour lui, une image est une zone graphique définie par un *handle* graphique, c'est-à-dire un numéro d'identification donnant accès à un tableau contenant toutes les informations sur l'image. Normalement, vous n'avez pas à vous préoccuper de ce genre de détail technique. C'est le *Visual Basic* qui s'occupe de gérer cela automatiquement. Mais dans ce cas précis, nous avons besoin des handles afin d'appeler la fonction graphique *BitBlt*. Heureusement, les concepteurs du *VB* ont prévu la méthode *hDC* pour obtenir le *handle* graphique d'une image.

```
LECTURE D'UN HANDLE GRAPHIQUE
HDestDC = Image1.hDC
```

Le paramètre *dwRop&* indique la manière dont *Windows* effectue le transfert du bloc de pixels. Cela peut être une simple copie ou il peut y avoir une «*fusion*» entre les pixels d'origine et les pixels de destination. C'est une fonctionnalité très puissante qui permet de réaliser des effets graphiques très intéressants. Mais pour l'instant, nous allons nous contenter de copier simplement le bloc de pixels. Nous reviendrons ultérieurement sur les possibilités de «*fusion*» des images. Pour indiquer à *Windows* que la copie doit se faire sans mélange de couleurs, il faut donner la valeur *&HCC0020* au paramètre *dwRop&*. C'est une constante système de *Windows* dont la valeur a été définie par les programmeurs de Microsoft.

La fonction *BitBlt* nécessite que les tailles des images soient exprimées en pixels. C'est gênant, car le *VB* utilise les twips comme unité de mesure pour indiquer la taille et la position des contrôles graphiques. C'est une unité de mesure virtuelle qui correspond à 576 twips par centimètre. Pour indiquer que les caractéristiques d'un contrôle sont exprimées en pixels, il faut modifier sa propriété *ScaleMode* de la manière suivante :

```
Const PIXEL = 3

Image1.ScaleMode = PIXEL
```

Il nous reste encore une chose à voir avant de passer aux exercices pratiques. Le transfert du bloc de pixel ne se fait correctement que si la propriété *AutoRedraw* de l'image de des-

Système : Windows 3.1 et Windows 95
Langage : Visual Basic 3.0

tination contient la valeur *TRUE*. Mais alors, le bloc n'apparaît pas sur l'écran après l'affichage ! Lorsque *AutoRedraw* est actif, le *VB* conserve en mémoire une copie de sauvegarde de l'image. Dans ce cas de figure, la propriété *hDC* fournit le *handle* de l'image de sauvegarde et non le *handle* de l'image présente sur l'écran. Le transfert *BitBlt* se fait alors dans la copie de sauvegarde et non sur l'écran. Pour restaurer le contenu de l'écran avec la copie, il faut utiliser la méthode *Refresh* après chaque utilisation de *BitBlt*. Cela se fait de la manière suivante :

```
Image2.Refresh
```

La bibliothèque BITBLT.BAS

Comme vous pouvez le constater, l'utilisation de *BitBlt* n'est pas très pratique ! C'est pourquoi nous avons écrit plusieurs routines graphiques simplifiant l'usage de *BitBlt* et permettant d'obtenir des effets graphiques amusants. Ces fonctions sont regroupées dans une bibliothèque portant le nom de *BITBLT.BAS*. Pour l'utiliser dans vos programmes, il suffit de charger le fichier *BITBLT.BAS* dans la fenêtre de projet.

La première de ces routines est *MoveImage* qui permet de transférer un bloc de pixels entre deux contrôles de types *PictureBox*. C'est une version améliorée de *BitBlt*, plus facile à utiliser que l'originale. Voici sa syntaxe :

```
Sub MoveImage (Image1 As PictureBox,
               Image2 As PictureBox,
               px1%, py1%,
               tx%, ty%,
               px2%, py2%)
```

```
Image1: contrôle
graphique d'origine
Image2: Contrôle gra-
phique de destination
px1%, py1%: Position
d'origine du bloc
tx%, ty%: Taille du
bloc
px2%,py2%: Position de
destination
```

▲ *Le programme Bitblt01.mak copie plusieurs fois le même bloc de pixels sur un contrôle graphique.*

Après le transfert, n'oubliez pas d'utiliser la méthode *Refresh* pour visualiser le résultat sur l'écran. Sinon, la nouvelle image restera cachée dans les entrailles de *Windows*.

```
CALL MoveImage(Image1,Image2, ..)
Image2.Refresh
```

Le programme *Bitblt01.mak* utilise cette routine pour copier plusieurs fois le même bloc de pixels sur un contrôle graphique de type *PictureBox*. Voici le coeur de ce programme :

```
px1% = 20
py1% = 20

px2% = 5
py2% = 5

For n = 1 To 8
  Call MoveImage(image1, image2,
                 px1%, py1%,
                 tx%, ty%,
                 px2%, py2%)
  px2% = px2% + 15
  py2% = py2% + 10
Next n
image2.Refresh
```

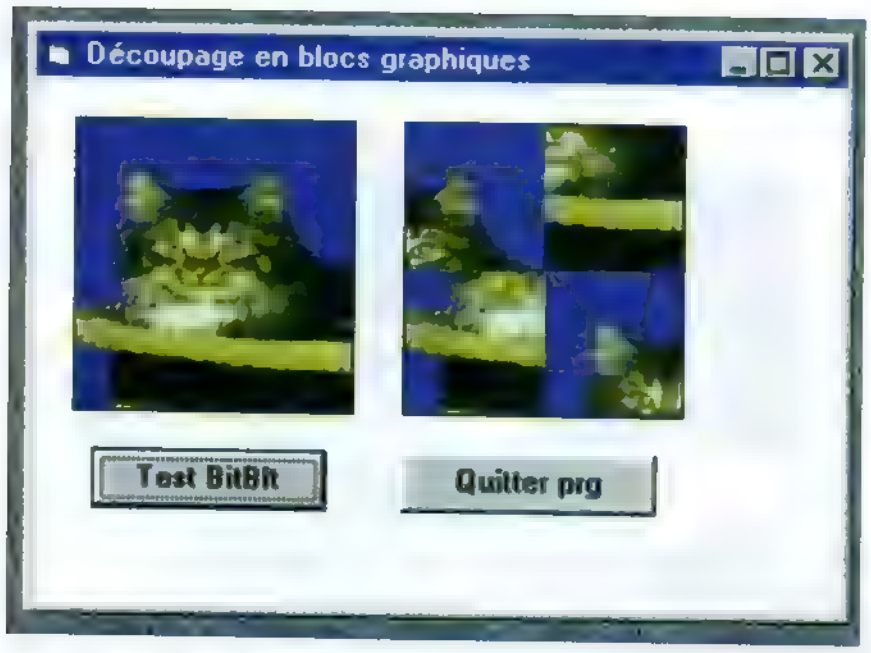
Cette routine prend un bloc graphique sur le contrôle *image1* et le copie à 8 endroits différents du contrôle *image2*. A chaque affichage, la position *x* varie de 15 pixels et la position *y* de 10 pixels. Cela produit un effet d'escalier assez intéressant.

La routine MoveBLOC

Certains jeux, comme le puzzle ou les jeux de mémoire, sont basés sur un découpage de l'écran, sous la forme de rectangles de même taille. Pour traiter cette technique, nous avons écrit une routine permettant de découper automatiquement une image en blocs rectangulaires. Sa syntaxe est la suivante :

```
Sub MoveBloc (Image1 As PictureBox,
              image2 As PictureBox,
              l1%, c1%, l2%, c2%,
              nb_lignes%,
              nb_colonnes%)

Image1 : Contrôle graphique d'origine
Image2 : Contrôle graphique de desti-
nation
l1%, c1% : Position d'origine du bloc
(ligne et colonne)
tx%, ty% : Taille du bloc en pixels
px2%,py2% : Position de destination
(ligne et colonne)
```

Cette routine copie un bloc du contrôle graphique *Image1* vers le contrôle graphique *Image2*. La taille du bloc est calculée automatiquement par la routine, à partir des dimensions du contrôle *Image1* et de la valeur des paramètres *nb_lignes* et *nb_colonnes*. Par exemple, si l'image fait 200x200 pixels et que le découpage se fait avec 10 lignes et 10 ▶



▲ *Le programme Bitblt02.mak est une démonstration de la routine MoveBloc qui découpe une image en plusieurs rectangles de même taille.*

colonnes, chaque bloc graphique occupera 20x20 pixels.

Le programme *bitblt02.mak* utilise cette routine pour découper une image en 4 éléments et afficher ces éléments dans le désordre. Ce peut être la base d'un jeu de taquin où il faut remettre les morceaux d'images dans le bon ordre. Comme vous pourrez le voir en chargeant le programme, l'éditeur de votre *VB*, la technique de découpage utilisée permet toutes les combinaisons de découpages possibles.

Les limitations de BitBlt

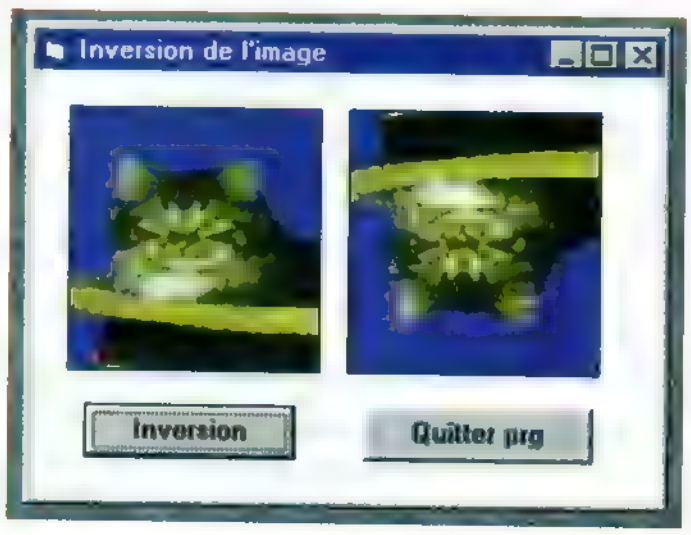
La fonction *BitBlt* est très pratique, mais elle souffre de quelques défauts qu'il est bon de connaître avant de l'utiliser intensivement. Pour commencer, elle ne peut fonctionner qu'avec des contrôles graphiques situés dans la même feuille. Si vous tentez de transférer des données graphiques entre feuilles différentes avec la fonction *BitBlt*, il va se passer des choses étranges. Par exemple, ce peut être le fond d'écran *Windows* qui va être copié. Plutôt bizarre ! Cela limite un peu l'intérêt de *BitBlt* ! Mais il est possible de contourner la difficulté, en transférant d'abord l'image à afficher dans un contrôle invisible (propriété *Visible* = *False*) et d'utiliser ensuite le transfert *BitBlt* entre le contrôle invisible et un contrôle graphique visible (propriété *Visible* = *True*). **Attention :** cela ne fonctionne que si la propriété *AutoRedraw* du contrôle invisible a la valeur *True* !

Il existe une autre limitation liée au nombre de couleurs du système d'affichage. Elle ne se manifeste que sur les versions de *Windows* configurées en 256 couleurs. Si c'est le cas de votre machine, *Windows* ne peut afficher que 256 couleurs différentes sur l'écran. Tout fonctionne correctement si vous avez un seul contrôle graphique à l'écran. Cela se complique si vous avez deux images sur la même feuille, dans deux contrôles graphiques. A première vue, les deux images sont parfaitement visibles, même si les palettes de couleurs sont différentes. En fait, *Windows* fait une cuisine interne complexe pour accorder subtilement les palettes de couleurs, de manière à ce que le spectateur ait l'impression de voir deux images différentes. Si vous projetez un bloc de pixels d'une image sur l'autre, il va se produire une bavure graphique. La seconde image va adopter la palette de la première image, ce qui produit généralement un effet graphique très déplaisant. Pour éviter cela, il faut utiliser des images ayant exactement la même palette de couleurs ou configurer *Windows* pour qu'il fonctionne dans un mode graphique avec beaucoup de couleurs. Les cartes graphiques actuelles peuvent fonctionner en 65 000 couleurs, sans perte de vitesse. Cela donne une qualité d'image supérieure au mode 256 couleurs !

Pour finir

Vous savez maintenant comment utiliser la fonction *BitBlt*. C'est une fonction importante qui permet de réaliser des effets graphiques intéressants. Vous pouvez l'utiliser pour écrire des jeux et des démonstrations graphiques. Vous pouvez également réaliser un système de cryptage graphique qui code une image en la découpant en petits carrés et en mélangeant ces éléments dans un ordre précis. Dans ce cas, la clé de décryptage est l'ordre inverse de mélange des petits carrés.

Jacques Lemercier



Une nouvelle routine

Vous trouverez sur le disque le programme BitBlt03.mak. Il inverse le sens d'affichage d'une image, en utilisant une routine permettant de copier une ligne d'un contrôle graphique vers un autre.

AVEZ-VOUS REMARQUÉ QUE LE QBASIC NE SAIT PAS COMMENT GÉRER LA SOURIS ? DE PLUS, IL NE POSSÈDE MÊME PAS DE MÉCANISME PERMETTANT DE CRÉER UN GESTIONNAIRE SOURIS. C'EST VRAIMENT DOMMAGE ! IL FAUT RUSER POUR DÉPASSER CETTE LIMITATION.

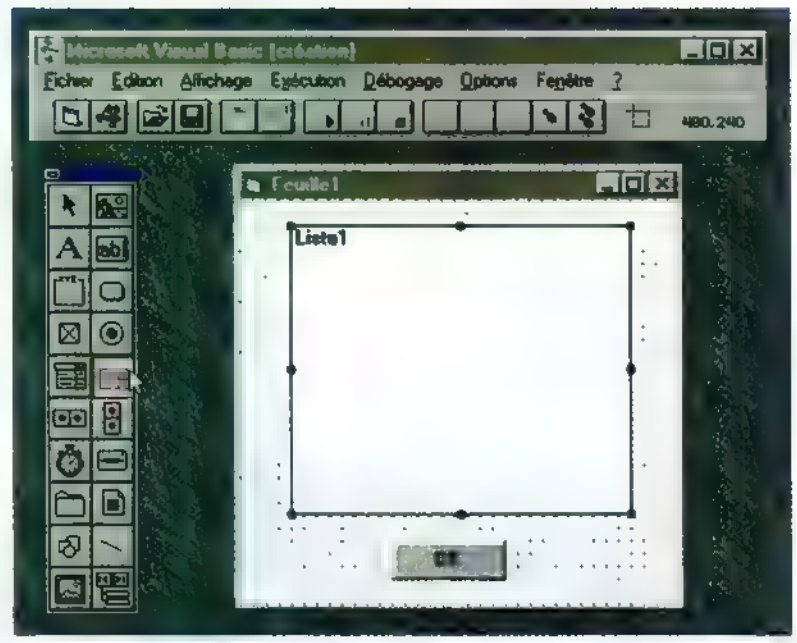
Trucs et astuces

Vous avez des problèmes avec le *Visual Basic* ? Vous n'arrivez pas à faire quelque chose de particulier ? Une instruction du *VB* vous pose un problème ? Rassurez-vous, nous sommes là pour vous aider !

Les ouvrages d'initiation au *Visual Basic* se consacrent surtout à la présentation des instructions au langage. Ce n'est pas suffisant car la programmation est surtout une affaire de «recettes de cuisine». Il existe des solutions simples pour la plupart des problèmes de programmation. Mais les débutants ne les connaissent pas et perdent beaucoup de temps à «réinventer la poudre». C'est pourquoi nous avons décidé de créer une rubrique ayant pour vocation de vous présenter certaines «recettes» de base.

Je veux afficher un texte dans une fenêtre, mais elle trop petite pour contenir tout le texte. Comment faire pour faire apparaître une barre de défilement verticale permettant de visualiser la totalité du texte ?

Les programmeurs qui utilisent les anciens basics du type Qbasic ou *Quick Basic* n'ont pas de problèmes pour afficher des chiffres ou des textes sur l'écran. Le seul moyen d'afficher quelque chose sur l'écran est d'utiliser l'instruction *PRINT*. C'est nettement plus compliqué avec le *Visual Basic* et ses nombreux contrôles graphiques. Il existe de nombreux moyens d'afficher des informations sur l'écran. Le plus classique est d'utiliser un contrôle *Etiquette* et de modifier son contenu, grâce à la propriété *Caption*. Il est également possible d'utiliser l'instruction *PRINT* pour écrire des informations dans un contrôle graphique de type *PictureBox* ou sur le fond d'une feuille. Mais cela ne nous explique pas comment utiliser une barre de défilement verticale pour visualiser la totalité d'un texte long !



▲ *Il suffit de cliquer sur l'icône correspondante pour créer un contrôle ListBox.*

Pour parvenir à notre objectif, il faut abandonner les techniques classiques d'affichage et utiliser un contrôle de type *ListBox*. Ce contrôle graphique permet de présenter à l'utilisateur une liste d'objets qu'il peut sélectionner (une liste de fichiers, par exemple). Pour afficher un texte, il suffit de remplir la liste avec les différentes lignes du texte. Le contrôle se charge ensuite de l'affichage du texte et de faire apparaître une barre de défilement si le texte est plus long que le contrôle. C'est très pratique à utiliser, puisque c'est le con-trôle qui s'occupe de tout ! Le plus difficile est de remplir la liste avec les lignes du texte. Cela se fait avec la méthode *Additem*. Le listing suivant vous montre comment remplir une liste avec un texte :

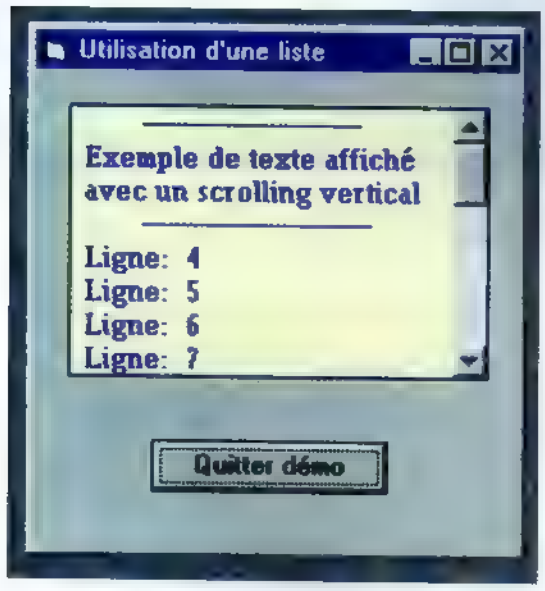
```
' EFFACEMENT CONTENU LISTE
Liste1.Clear
' REMPLISSAGE LISTE
For n = 0 To 30
  ligne$ = «Ligne: « + Str$(n)
  liste1.AddItem ligne$
Next n
```

La méthode *Clear* permet d'effacer le contenu antérieur de la liste avant de la remplir avec les lignes du texte. Le remplissage se fait avec la méthode *Additem* qui ajoute un élément à la liste, chaque fois qu'elle est utilisée. La fonction *Str$* permet de convertir le numéro de la ligne en un texte qui est ensuite ajouté au texte «*Ligne*». Si vous utilisez cette technique pour afficher un texte important, il est probable que celui-ci soit stocké dans un tableau. Dans ce cas, le programme d'affichage doit lire ce tableau pour remplir la liste. Cela se fait de la manière suivante :

```
liste1.Clear
For n = 0 To nb_lignes
  liste1.AddItem t$(n)
Next n
```

C'est très facile à mettre en œuvre. Le plus difficile est de remplir le tableau avec le texte. Cela peut se faire dans le programme de la manière suivante :

```
ReDim t$(30)

t$(0) = «————————»
t$(1) = «Affichage d'un texte»
t$(2) = «avec une barre de»
t$(3) = «défilement verticale»
t$(4) = «————————»
t$(5) = ..
t$(6) = ..
nb_lignes=10
```



▲ *Le programme scroltxt.mak affiche un texte dans une fenêtre avec une barre de défilement verticale.*

Vous trouverez sur le disque le programme *scroltxt.mak* qui utilise cette technique. Il est écrit en *Visual Basic 3.0* et fonctionne aussi parfaitement en *VB 4.0*. Si la manipulation des listes vous intéresse, vous pouvez étudier les instructions *RemoveItem*, *ListIndex*, *List* et *ListCount*. L'aide intégrée du *VB* contient des informations intéressantes sur ces instructions.

Comment puis-je faire pour utiliser des sons dans un programme en *VB*?

La seule instruction sonore standard du *Visual Basic* est l'instruction *Beep* qui fait retentir un signal sonore plus ou moins fort. Il est dommage que les concepteurs du *VB* n'aient pas pensé à prévoir une utilisation directe des fichiers *WAVE*. Ce type de fichier est pourtant incontournable, puisque *Windows* l'utilise dans son système multimédia.

Heureusement, le système d'exploitation de *Windows* contient une fonction toute prête pour jouer des fichiers *WAVE*. C'est la fonction snd*PlaySound()* qui se trouve dans le fichier *DLL MMSYSTEM*. Il est possible d'utiliser une fonction *DLL* dans un programme *VB*, à condition de déclarer son nom, sa *DLL* d'origine et sa syntaxe. Voici la formulation à utiliser pour *sndPlaySound()*:

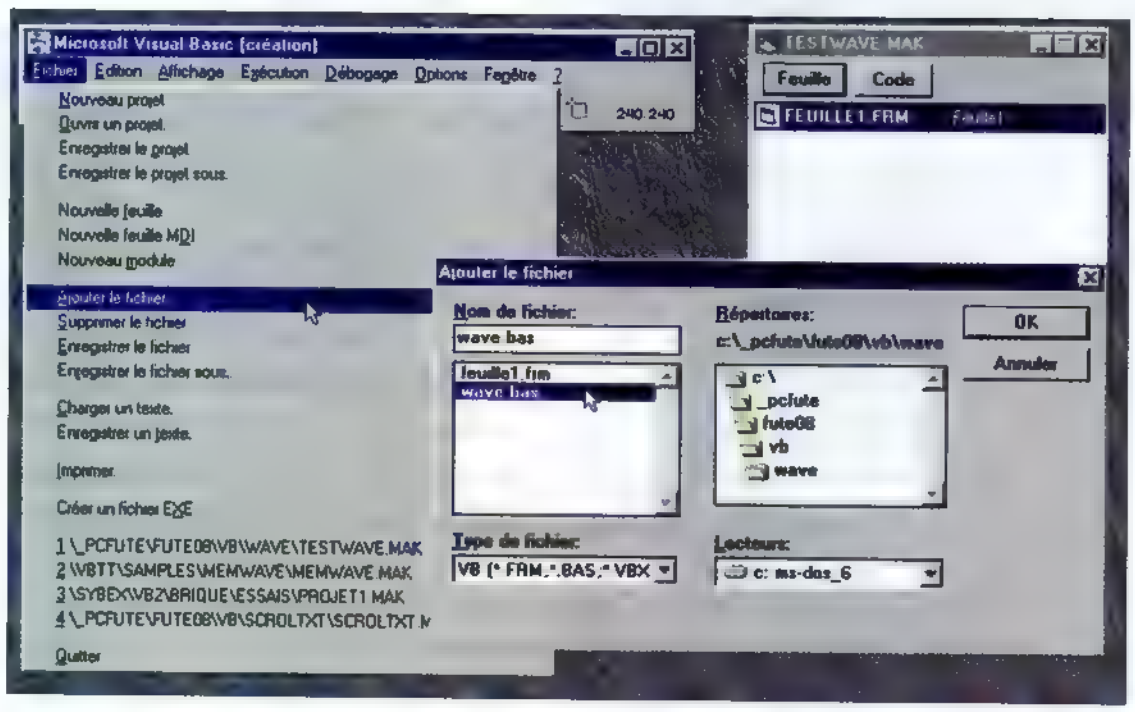
```
Declare Function sndPlaySound
        Lib «MMSystem»
       (ByVal lpsound As String,
        ByVal flag As Integer)
        As Integer
```

La déclaration doit se faire dans la partie (déclarations) d'une feuille ou d'un module *BAS*. Le premier paramètre est une chaîne contenant le nom du fichier *WAVE*. Le nom doit contenir le chemin complet (par exemple *«c:\vb\tests\bruit.wav»*). Le second paramètre indique la manière dont le son doit être restitué. La *valeur 1* correspond à une restitution normale.

Le fait de devoir indiquer le répertoire du fichier *WAVE* est très contraignant. Cela oblige à stocker tous les sons dans le même répertoire ou d'indiquer en dur dans le logiciel la position du fichier (très mauvaise idée !). Cependant, il existe une solution plus astucieuse. L'objet *APP* contient différentes informations sur le programme courant. Par exemple, la propriété *Path* contient le répertoire d'appel du logiciel et le répertoire d'appel du logiciel *«c:\vb\tests\prg1.exe»* est le répertoire *«c:\vb\tests»*. Si les fichiers au format *WAVE* sont placés dans le même répertoire que le programme, vous pouvez utiliser la propriété *APP.Path* pour connaître leurs répertoires.

```
Sub JouerWave (NomFichier As String)
 Dim rep As Integer
 Dim nom As String

 nom = APP.Path + «\» + NomFichier
 rep = sndPlaySound(nom, 1)
End Sub
```

La fonction *JouerWave()* permet de jouer un fichier *WAVE* contenu dans le même répertoire que le programme principal. Elle est écrite à partir de la fonction *sndPlaySound()* et



▲ *Pour insérer un module BAS dans un projet, il faut ouvrir le menu «Fichier», sélectionner la fonction «Ajouter un fichier...», puis utiliser le sélecteur de fichier pour indiquer au VB le nom du module à charger.*

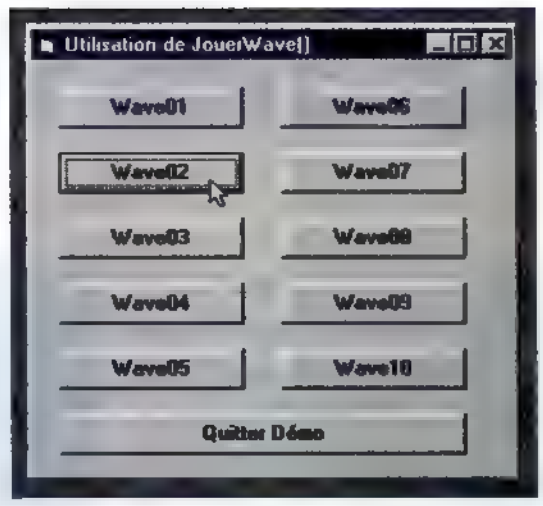
utilise la propriété *APP.Path* pour déterminer la position du fichier sonore.

Afin de simplifier l'utilisation des fonctions sonores, nous avons placé les fonctions *sndPlaySound()* et *JouerWave()* dans un module *BAS* portant le nom de *WAVE.BAS*. Si vous voulez utiliser ces fonctions dans un programme, il faut copier le fichier *WAVE.BAS* dans le même répertoire que le programme et l'intégrer dans le projet, grâce à la fonction «*Ajouter le fichier...*» du menu «*Fichier*». Cela vous permet d'utiliser notre module *WAVE.BAS* dans vos propres programmes, sans retaper le code des fonctions.

Le programme *TestWave* que vous trouverez sur la disquette est un exemple d'utilisation de la fonction *JouerWave()*. Il utilise le module *WAVE.BAS* et permet d'écouter le son d'une dizaine de fichiers *WAVE* présents sur la disquette. Le logiciel est très simple, puisqu'il se compose uniquement de boutons de commande. Chaque pression sur un bouton provoque l'émission d'un son particulier. Voici le code correspondant au premier bouton :

```
Sub Commande1_Click ()
  Call JouerWave(«wave01.wav»)
End Sub
```

Vous pouvez vous procurer des fichiers *WAVE* sur les disquettes et les CD-ROM du domaine public. Il y a de tout, de la musique la plus sublime aux bruitages les plus répugnants ! Vous utilisez les fichiers courts pour illustrer une image ou attirer l'attention sur une boîte d'alerte. Rien n'est plus efficace qu'une sirène d'incendie pour motiver l'utilisateur le plus distrait ! **Attention** : les enregistrements sonores prennent beaucoup de place. C'est pourquoi il est préférable d'utiliser des fichiers de petite taille.



▲ *Le programme Testwave permet d'écouter les sons de 10 fichiers WAVE placés sur la disquette.*

J'ai quelques problèmes pour passer du *Quick Basic* au *Visual Basic 3.0*. Par exemple, je ne parviens pas à utiliser l'instruction *Dim* pour créer un tableau alphanumérique dont le contenu est utilisable par toutes les feuilles de mon programme. Comment faire ?

Pour créer un tableau accessible par toutes les feuilles d'un programme, il faut déclarer un module *BAS* et utiliser l'instruction *Global* pour créer le tableau.

```
Global Nom$(10)
```

L'instruction *Global* indique au *VB* que le tableau *Nom$* peut être utilisé par toutes les fonctions de toutes les feuilles du programme. Cette instruction ne peut être utilisée que dans la partie (déclarations) d'un module *BAS*. Si vous utilisez l'instruction *Dim* à la place de *Global*, le tableau n'est défini que pour les routines du module *BAS*.

```
Global Const NB_IMAGES = 20
```

Cette technique peut également être utilisée pour créer des constantes globales, utilisables par la totalité du programme.

Comment puis-je faire pour créer une bibliothèque de fonctions *VB* utilisables dans plusieurs programmes ?

Un programme en *VB* est constitué de plusieurs fichiers contenant chacun un élément particulier du programme. Les fichiers *FRM* contiennent la définition des feuilles et les fichiers *BAS* contiennent des routines en *Visual Basic*. Le principe de base de la programmation en *VB* est que le corps du programme est écrit dans un fichier *BAS*, tandis que les routines des feuilles s'occupent uniquement de la communication avec l'utilisateur. Le *VB* peut utiliser plusieurs fichiers *BAS* en même temps. Cela permet de créer des bibliothèques de fonctions spécialisées, en mettant toutes les routines de même type dans un fichier *BAS* spécifique. C'est ce que nous avons fait plus haut avec le fichier *WAVE.BAS*, où toutes les routines sonores sont regroupées dans un seul fichier *BAS* qu'il est facile d'utiliser dans n'importe quel projet.

Si une bibliothèque de fonctions stockée dans un fichier *BAS* a besoin d'une variable pour stocker des données, il faut déclarer cette variable avec l'instruction *Dim*. Cela indique au *VB* que la variable est une variable globale définie uniquement pour ce module. C'est une sorte d'intermédiaire entre une variable globale accessible depuis la totalité du programme (déclaration avec *Global*) et une variable locale à une routine (déclaration avec *Dim* au début de la routine). Cela permet d'utiliser des variables globales dans un fichier *BAS*, sans se préoccuper du fait qu'un autre fichier *BAS* risque d'utiliser le même nom de variable. Les concepteurs du *VB* ont imaginé ce système pour faciliter la création de bibliothèques de fonctions dans des fichiers *BAS* indépendants les uns des autres.

Mike Terranova

LA DISQUETTE DU MOIS CONTIENT PLUSIEURS FICHIERS POUR LE VISUAL BASIC 3.0 ET 4.0. IL Y A DES PROGRAMMES D'EXEMPLES À ÉTUDIER ET DES EXTENSIONS VBX INTÉRESSANTES. IL Y A DE QUOI S'AMUSER PENDANT PLUSIEURS HEURES !

Les logiciels VB

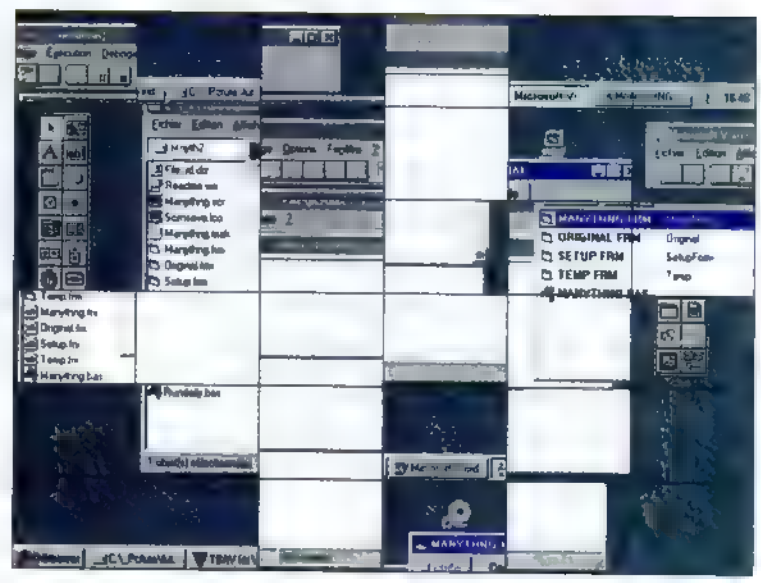
Deal2

Deal2 est un petit programme qui détermine aléatoirement des mains de 5 cartes à partir d'un jeu de 52 cartes. Il est écrit en *VB 3.0* et fonctionne parfaitement en *VB 4.0*. Les images BMP des 52 cartes sont stockées dans un ensemble de contrôles graphiques, situé sur la seconde feuille de l'application.

▲ *Ce jeu de 5 cartes a été tiré aléatoirement par le programme Deal2.*

Mnyth2

Le programme *Mnyth2* est un économiseur d'écran écrit entièrement en *VB 3.0*. Il peut générer plus d'une vingtaine d'effets graphiques amusants, directement sur le fond d'écran *Windows* (scrollings, effacements partiels, recomposition aléatoire de l'écran, affichage d'éléments graphiques en superposition, etc.). C'est un programme intéressant à étudier pour mieux comprendre le fonctionnement du graphisme en *VB*. La conversion en *VB 4.0* nécessite un peu de travail de réécriture.



▲ *L'économiseur d'écran Mnyth2 produit des effets graphiques intéressants, directement sur le fond d'écran Windows !*

Bvs-ssav

Ce programme au nom curieux est un économiseur d'écran écrit spécifiquement pour le *VB 4.0*. Il est plus simple que le précédent et ne compte qu'un seul effet graphique. C'est un exemple d'utilisation intéressant de la fonction *Bitblt()* que nous présentons dans un autre article du magazine.

Tbexit

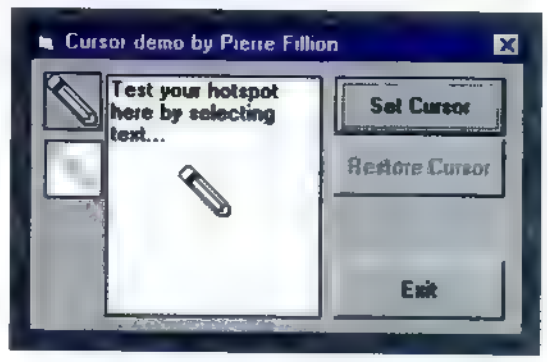
Cet utilitaire écrit en *VB 4.0* implante un programme dans la liste d'icônes de *Windows 95*. C'est une barre d'options qui permet de quitter le système, sans passer par le bouton «*Démarrer*». Vous pouvez étudier cette technique pour implanter vos propres utilitaires dans la barre d'icônes.

Cursor

Ce programme permet de changer la forme du curseur souris par une image de 32x32 pixels, ce qui est très intéressant pour le *VB 3.0*. En revanche, cela ne sert à rien en *VB 4.0*, puisque celui-ci est capable d'utiliser n'importe quelle icône comme curseur souris.

Bigfoot

Ce programme permet de visualiser à l'écran le contenu d'un fichier texte. Le texte est chargé dans un contrôle *TextBox* et une barre verticale permet de visualiser la totalité du texte. Le programme est écrit en *VB 3.0* et nécessite quelques petites modifications pour fonctionner en *VB 4.0*.



▲ *Le programme Cursor permet de remplacer le curseur souris par une image de 32x32 pixels. Ici, la nouvelle souris est un crayon de papier.*

Blist

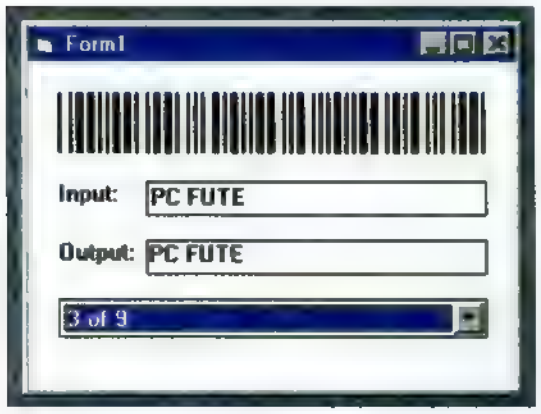
Le contrôle *Listbox* du *VB* permet de créer une liste d'objets de taille quelconque. Lorsque la liste dépasse les limites du contrôle, le système affiche une barre de défilement verticale. Le contrôle *PCX Blist* est une version améliorée de *Listbox* qui affiche une barre de défilement horizontale lorsqu'un élément de la liste est trop large pour tenir dans les limites du contrôle.

3Dinput

Ce logiciel est la démonstration d'un contrôle *VBX* permettant d'afficher un texte dans une étiquette. L'affichage et le dessin de l'étiquette peuvent se faire avec des effets 3D.

Barcode

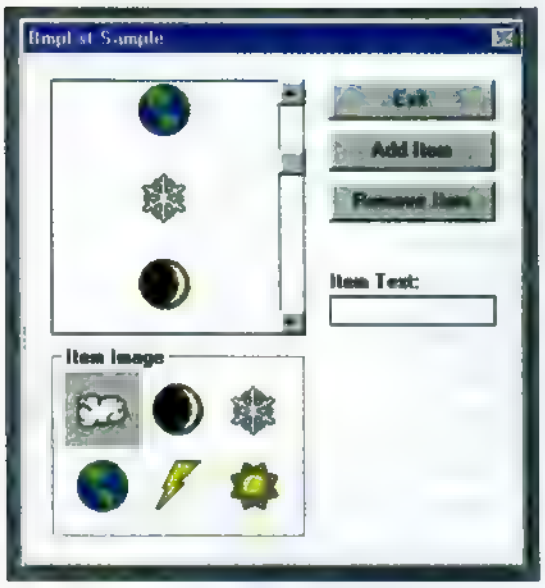
Ce logiciel est un programme de démonstration utilisant un *VBX* spécial pour afficher un code barres. Vous pouvez utiliser ce *VBX* pour générer vos propres codes barres. Le logiciel est capable d'utiliser les 4 formats de codes barres les plus courants.



▲ *Le programme Codebare peut fabriquer des codes barres à la demande.*

Bmplist

Ce contrôle *VBX* permet de stocker des images dans une liste. Si le nombre d'images est trop important, le programme affiche automatiquement une barre verticale permettant de scroller sur l'ensemble de la liste.



▲ *Le contrôle VBX Bmplist permet de stocker des images dans une bibliothèque graphique.*

Compatibilité VB 4.0

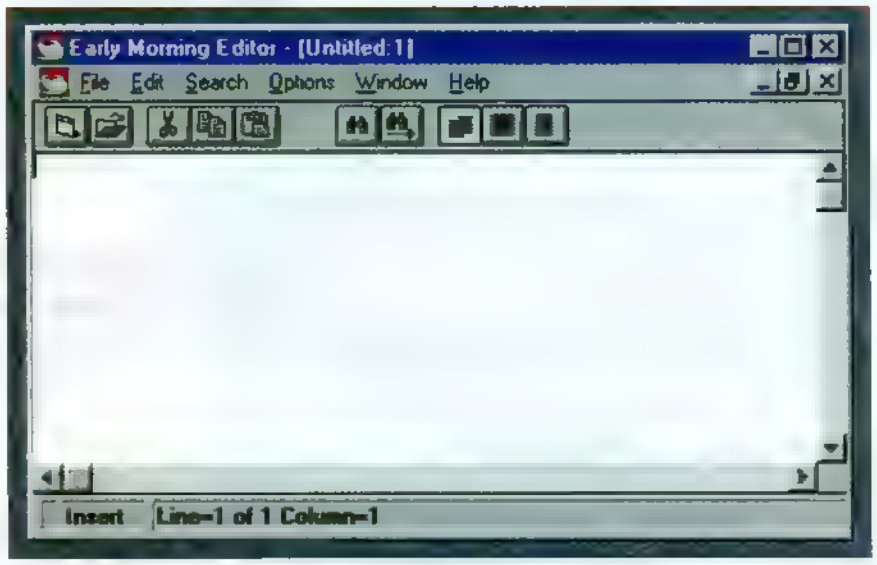
Le VB 4.0 est conçu pour utiliser des extensions 32 bits OCX. Il ne peut utiliser les extensions VBX du VB 3.0. C'est pourquoi les programmes utilisant un VBX ne peuvent être adaptés au VB 4.0, à moins de posséder un OCX équivalent au VBX.

Dinfo

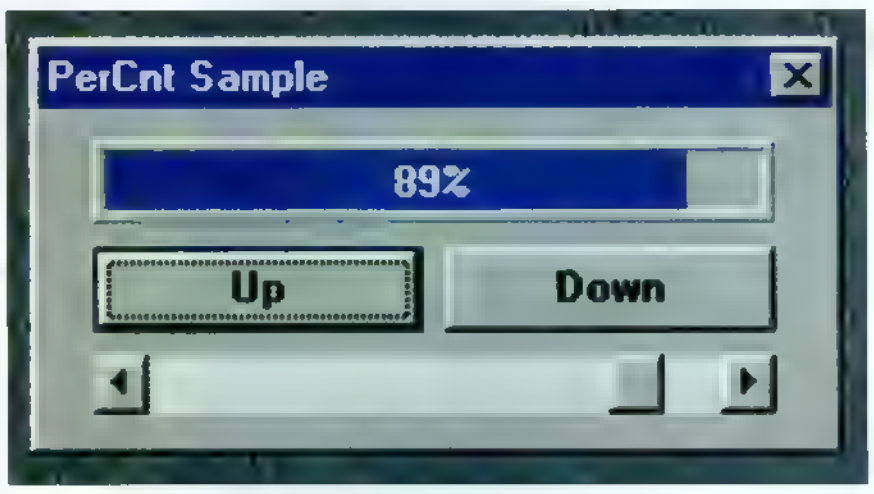
Le contrôle *VBX Dinfo* permet d'obtenir des informations sur un disque ou un fichier. Par exemple, il peut indiquer la place disponible sur un disque. Ou encore, indiquer l'état des attributs systèmes d'un fichier (*Lecture/Ecriture, Archive, Date de création, Taille,* etc..). Cela peut être très utile dans certaines applications faisant beaucoup appel aux fichiers.

Emedit

Ce programme est un éditeur de texte écrit autour d'un contrôle *VBX* capable de contenir un texte complet. Le source complet de l'éditeur est fourni. Ce système vous permet d'intégrer un véritable éditeur de texte dans vos propres programmes.



▲ *Le contrôle VBX Emedit permet d'intégrer un éditeur de texte dans le corps de n'importe quel programme écrit en VB 3.0.*



▲ *Le contrôle Percent affiche une barre de pourcentage indiquant l'état d'avancement d'une tâche.*

Percent

Le contrôle *VBX Percent* permet de créer une barre de pourcentage qui se remplit au fur et à mesure de l'exécution d'une tâche. Cela évite à l'utilisateur de regarder un écran statique en se demandant si le programme exécute bien ses ordres ou si il est planté !

Remarque

Certains contrôles VBX présentés ici sont des sharewares, c'est-à-dire des versions d'évaluation. Si vous les utilisez régulièrement, vous avez le devoir moral de payer la licence d'utilisation aux auteurs. Cette petite somme d'argent leur permet de vivre et de créer d'autres logiciels.

Jacques Lemercier

LE MOMENT TANT ATTENDU EST VENU! NOUS ALLONS DONNER L'ÉTINCELLE DE VIE À LA CRÉATURE ET LA FAIRE BOUGER. LE DONJON VA ENFIN AVOIR UN GARDIEN ET LES ENVAHISSEURS VONT SOUFFRIR! VAS-Y, IGOR, ENVOI LES 100.000 VOLTS DANS LA CRYPTE!

Créez votre Pacman (III)

Dans les précédents numéros de PC FUTE nous avons vu comment créer un labyrinthe et comment gérer les déplacements du pacman. Nous abordons aujourd'hui la phase terminale avec le déplacement des fantômes. Afin de simplifier la programmation, nous n'allons programmer qu'un fantôme. Ce n'est pas gênant, car la technique de déplacement est la même, quel que soit le nombre d'ectoplasmes!

Un fantôme est une créature qui se balade dans le labyrinthe et se jette sur le pacman quand il est à proximité. Comment programmer cela ? Le plus difficile dans la création d'un logiciel de simulation est de définir le comportement d'un objet ou d'une créature en termes informatiques. La première chose à faire est de définir avec précision l'environnement de jeu.

Dans le cas du pacman, l'environnement de jeu est une série de couloirs partant dans toutes les directions. Les déplacements ne peuvent se produire que dans le sens vertical ou horizontal. Cela ne représente que 4 directions possibles (haut/bas, droite/gauche). Le fantôme doit se déplacer dans cet environnement et foncer sur le pacman dés qu'il le voit. Cela représente deux comportements distincts: la promenade plus ou moins aléatoire et la chasse.

Nous allons commencer par étudier la promenade. Le fantôme se déplace au hasard dans le labyrinthe. La méthode la plus simple pour programmer cela est de considérer que le fantôme fonce tout droit devant, jusqu'au moment où il rencontre un obstacle. Chaque fantôme doit posséder une mémoire rudimentaire lui rappelant sa direction de déplacement et un système de vision simple indiquant la présence d'un mur. La direction peut être conservée dans une variable globale. Il suffit de définir une convention de direction (1=haut; 2=bas; 3=gauche; 4=droite).

```
Global direction_fantome
       As Integer
```

Le système de vision est plus complexe, car il faut que le programme soit capable de «lire» dans la mémoire de l'ordinateur afin de déterminer l'absence ou la présence d'un mur. Comme nous l'avons vu dans les articles précédents, l'espace de jeu est mémorisé dans le tableau *Tableau*, dont voici la définition:

```
Global tableau(1 To NB_LIGNES,
               1 To NB_COLONNES)
               As Integer
```

Chaque case du tableau correspond à une pièce du labyrinthe. Il peut y avoir 10 éléments différents dans une pièce. Voici la liste de ces objets:

```
Global Const MUR = 0
Global Const PACK_GOMME = 1
Global Const PACK_GOMME_ROUGE = 2
Global Const SOL = 3
Global Const PACKMAN = 4
Global Const PACKMAN_ROUGE = 5
Global Const FANTOME_ROUGE = 8
Global Const FANTOME_VERT = 6
Global Const FANTOME_BLEU = 7
Global Const BONUS = 9
```

Chaque objet est identifié par une constante symbolique. Pour savoir si une pièce du labyrinthe est occupée par un mur, il suffit de regarder le contenu de la case correspondante dans le tableau *Tableau*. C'est ce que fait la fonction *test_deplacement()*. Elle retourne la valeur *True* si le fantôme peut aller dans la case (*l*, *c*), et *False* dans le cas contraire. En fait, la routine n'autorise le déplacement que si la case est vide (constante *SOL*), ou ne contient qu'un pack gomme (constante *PACK_GOMME*)

```
Function test_deplacement (l, c)
 Dim n As Integer
 Dim rep As Integer

 rep = False
 n = tableau(l, c)
 If n = SOL Then rep = True
 If n = PACK_GOMME Then
   rep = True
 end if
 test_deplacement = rep
End Function
```

Cette routine est trop restrictive. Elle interdit le déplacement si la case contient un bonus ou un pack gomme rouge. Si vous trouvez que c'est injuste pour le fantôme, vous pouvez modifier la routine pour étendre les autorisations de déplacement.

▲ *Le fantôme vient de voir le pacman! Notre héros arrivera-t-il à échapper à son terrible ennemi ?*

Affichage du fantôme

La position courante du fantôme est définie par les variables *ligne_fantome* et *colonne_fantome*. Ce sont des variables définies avec l'attribut *Global*, de manière à être utilisables par toutes les routines du programme.

```
Global ligne_fantome As Integer
Global colonne_fantome As Integer
```

La fonction *afficher_fantome()* permet d'afficher le fantôme à sa position courante, définie par *ligne_fantome* et *colonne_fantome*. Pour cela, elle écrit la constante *IMAGE_FANTOME* dans la case correspondante du tableau *Tableau* et utilise la fonction *aff_case()* pour afficher la nouvelle image sur l'écran. Nous avons vu le fonctionnement de *aff_case()* dans le dernier PC FUTE. La constante *IMAGE_FANTOME* contient la valeur 8, c'est le numéro de l'image du fantôme rouge.

```
Sub afficher_fantome ()
  DIM l as integer
  DIM c as integer

  l = ligne_fantome
  c = colonne_fantome
  tableau(l, c) = IMAGE_FANTOME
  Call aff_case(l, c)
End Sub
```

Cette routine présente un défaut. Elle écrit le code du fantôme dans le tableau sans se préoccuper de son contenu antérieur. C'est gênant dans la mesure où le fantôme peut passer au-dessus de plusieurs types d'objets. Si l'information n'est pas conservée, le programme ne saura pas quoi réafficher lors du prochain mouvement du fantôme. Pour éviter cela, le logiciel doit sauver le contenu du tableau dans une variable globale. C'est ce que fait la routine suivante:

```
Sub afficher_fantome ()
  DIM l as integer
  DIM c as integer

  l = ligne_fantome
  c = colonne_fantome
  ' MEMORISATION PIECE FANTOME
  piece_fantome = tableau(l, c)
  tableau(l, c) = IMAGE_FANTOME
  Call aff_case(l, c)
End Sub
```

Effacement du fantôme

La routine *effacer_fantome()* efface l'image courante du fantôme, située à la position *ligne_fantome*, *colonne_fantome*. En fait, elle se contente de restaurer l'ancien contenu de la case, stocké dans la variable globale *piece_fantome*.

```
Sub effacer_fantome ()
 DIM l as integer
 DIM c as integer

 l = ligne_fantome
 c = colonne_fantome
 ' Restaure ancienne position
 tableau(l, c) = piece_fantome
 Call aff_case(l, c)
End Sub
```

Pour déplacer le fantôme d'une position à une autre, il faut effacer son image avec la fonction *effacer_fantome()*, modifier le contenu des variables *ligne_fantome* et *colonne_fantome* et appeler la routine *afficher_fantome()*.

```
effacer_fantome
ligne_fantome = ..
colonne_fantome = ..
afficher_fantome
```

Gestion du déplacement

Le fantôme doit se déplacer tout le temps dans la même direction, jusqu'à ce qu'il soit arrêté par un obstacle. La manière la plus simple de programmer cela est d'utiliser une boucle infinie. La routine calcule la position probable du fantôme et utilise la fonction *test_deplacement()* pour voir si le déplacement est possible. En cas d'obstacle, le programme détermine aléatoirement une nouvelle direction de déplacement et recommence les calculs. Sinon, le logiciel sort de la boucle *do loop* qui contient le calcul de la position et effectue le déplacement.

```
' GESTION DU DEPLACEMENT
Do
 Select Case direction_fantome
 ' ALLER HAUT
```

```
Case 1:
 ligne = ligne_fantome - 1
 colonne = colonne_fantome
' ALLER BAS
 Case 2:
  ligne = ligne_fantome + 1
  colonne = colonne_fantome
' ALLER GAUCHE
Case 3:
 ligne = ligne_fantome
 colonne = colonne_fantome - 1
' ALLER DROITE
Case 4:
 ligne = ligne_fantome
 colonne = colonne_fantome + 1
End Select
'
If test_deplacement(
          ligne,colonne) Then
  Exit Do
End If

direction_fantome=Int(Rnd*4+1)
Loop

' DEPLACEMENT FANTOME
effacer_fantome
ligne_fantome = ligne
colonne_fantome = colonne
afficher_fantome
```

Destruction du pacman

Si le fantôme se trouve à coté du pacman, il lui saute dessus et le détruit. Il faut intégrer un système de détection du pacman dans la routine de gestion du fantôme. C'est simple à réaliser dans la mesure où le fantôme ne peut attaquer le pacman que s'il se trouve sur la même ligne ou la même colonne, et à une distance d'une case seulement. Il suffit de comparer la position du fantôme et du pacman pour connaître la distance qui les sépare.

```
' Calcul distance pacman
d_ligne = ligne_fantome
          - ligne_packman
d_colonne = colonne_fantome
            - colonne_packman

If ((d_ligne = 0) And
   (Abs(d_colonne) = 1)) Then
       mort_packman
       Exit Sub
End If

If ((d_colonne = 0) And
   (Abs(d_ligne) = 1)) Then
      mort_packman
      Exit Sub
End If
```

La destruction du pacman est traitée par la routine *mort_packman*. Elle affiche l'image du fantôme à la place du pacman, émet un signal sonore, bloque tous les timers du jeu et affiche une boîte d'alerte avec le message «*Perdu*».

Un peu d'intelligence pour le fantôme

Le fantôme a peu de chance d'attraper le pacman si ces déplacements sont dus entièrement au hasard. Il faut lui donner un mode «chasse» qui augmente ces chances. La méthode la plus simple est de donner une «vision artificielle» au fantôme. Avant chaque déplacement, notre ectoplasme jette un coup d'oeil autour de lui. Si le pacman est dans le champ de vision, le fantôme change sa direction de déplacement pour foncer directement sur son gibier.

```
Function test_vision_bas ()
 Dim ligne As Integer
 Dim colonne As Integer
 Dim n As Integer
 Dim rep As Integer
 '
 ligne = ligne_fantome
 colonne = colonne_fantome
 rep = False
 Do
  n = tableau(ligne, colonne)
  If n = PACKMAN Then
    rep = True
    Exit Do
  End If
  If n = MUR Then
    Exit Function
  End If
  ligne = ligne + 1
 Loop
 '
 test_vision_bas = rep

End Function
```

La fonction *test_vision_bas()* vous montre comment fonctionne la vision du fantôme. Elle part de la position courante du fantôme et balaye le tableau de jeu vers le bas, jusqu'à rencontrer un mur ou le pacman. Elle retourne la valeur *True* si le pacman est détecté, et la valeur *False* si un mur bloque la vision. Il existe une routine pour chaque type de vision (haut, bas, droite et gauche). C'est la fonction *vision_fantome()* qui supervise ces routines afin de déterminer la nouvelle direction de déplacement.

```
Sub vision_fantome ()
 If test_vision_haut() Then
  direction_fantome = 1
  Exit Sub
 End If
 '
 If test_vision_bas() Then
  direction_fantome = 2
  Exit Sub
 End If
 '
 If test_vision_gauche() Then
  direction_fantome = 3
  Exit Sub
 End If
 '
 If test_vision_droite() Then
  direction_fantome = 4
  Exit Sub
 End If

End Sub
```

suite p. 61

Accès direct à la mémoire vidéo

LES JEUX MODERNES SONT DE PLUS EN PLUS RAPIDES. POUR OBTENIR CES PERFORMANCES, IL FAUT PLONGER DANS LES ENTRAILLES DU SYSTÈME ET ÉCRIRE DIRECTEMENT DANS LA MÉMOIRE VIDÉO. HEUREUSEMENT, LA STRUCTURE MÉMOIRE DU MODE GRAPHIQUE VGA 320X200 EST FACILE À MANIPULER !

La mémoire vidéo VGA 320x200 occupe exactement 64 000 octets. Elle est organisée en 200 lignes de 320 octets. Chaque octet contient le codage d'un pixel. Le premier octet se trouve à l'adresse A000:0000. Cela correspond au segment A000 et à l'offset 0000. Ce système de segment d'offset vient de la manière archaïque dont le *DOS* gère la mémoire. Elle est découpée en segments de 64 Ko. Pour connaître la position d'une donnée dans la mémoire, il faut connaître son segment et sa position par rapport au début du segment, c'est-à-dire son offset.

Accès à la mémoire vidéo

Pour manipuler le contenu de la mémoire vidéo, il faut pouvoir lire et écrire des données à n'importe quel endroit de la mémoire. C'est une tâche sur mesure pour les pointeurs du *C*. Un pointeur est un objet qui pointe sur une adresse précise de la mémoire. Par son intermédiaire, il est possible d'agir sur le contenu de cette adresse. Selon le type de pointeurs, on peut agir directement sur des octets, des entiers 16 bits, des chaînes de caractères, etc.

Le langage *C* ne possède pas de type de données correspondant à un octet. Mais il est possible d'en créer, grâce à l'instruction typedef. Le listing suivant vous montre comment créer un pointeur :

```
typedef unsigned char byte;

byte *ptr;
```

Ce pointeur n'est pas initialisé et peut pointer sur n'importe quel emplacement de la mémoire. Pour l'utiliser, il faut lui indiquer sur quelle adresse il doit pointer. C'est la tâche de la fonction *MK_FP()*. Elle reçoit un segment et un offset et génère le pointeur correspondant. Les adresses systèmes PC sont toujours exprimées en hexadécimal. C'est pourquoi nous avons écrit 0xA000 et 0x000. Le fait de commencer un nombre par 0x indique au *C* qu'il s'agit d'une valeur hexadécimale.

```
ptr = MK_FP(0xA000,0x000);
```

Il est probable que cette formulation provoque un message d'erreur sur votre compilateur. Cela vient du fait que *MK_FP()* est conçu pour fournir un pointeur de type indifférencié (*void **) et non un pointeur sur un octet (*byte **). Les compilateurs *C* vérifient que le type des données écrites dans une variable soit de même type que la variable. Sans cela, il y a un message d'avertissement. Pour éviter cela, il faut indiquer au compilateur qu'il doit convertir le résultat de *MK_FP()* en un pointeur sur un byte. Voici la syntaxe à utiliser :

```
ptr = (byte *)MK_FP(0xA000,0x0000);
```

Il est possible que vous ayez encore une erreur de compilation sur cette ligne. Cela provient du mode mémoire utilisé par votre compilateur. A cause des segments et des offsets, les compilateurs *C* possèdent plusieurs modes de gestion de la mémoire (*Small, Medium, Compact, Large* et *Huge*). Cela correspond à des manières plus ou moins ésotériques de gérer la mémoire. Le mode mémoire le plus pratique pour nous est le mode *Large*. Avant de compiler vos programmes, il faut activer l'option «*Mode Mémoire Large*» de votre compilateur.

Attention

N'oubliez pas d'activer le mode de mémoire «Large» avant de compiler vos programmes. Cela doit se faire généralement lors de la création du projet de compilation. Sans cela, vous aurez des problèmes pour créer un pointeur permettant d'accéder à la mémoire vidéo. Les compilateurs C ont généralement un panneau regroupant toutes les options de compilation.

La disquette

La disquette contient plusieurs programmes d'exemples écrits en Visual C et Borland C. Ils montrent que l'on peut faire des choses amusantes en manipulant directement la mémoire écran.

Lecture de la mémoire vidéo

Nous savons maintenant comment créer un pointeur sur la mémoire. Il ne reste plus qu'à l'utiliser pour accéder au contenu de la mémoire vidéo. C'est très simple car la syntaxe d'utilisation des pointeurs n'a rien de très compliqué. Voici la formule à utiliser pour lire le contenu de la mémoire adressée par un pointeur :

```
byte  valeur;
byte *ptr;

valeur = *ptr;
```

C'est l'opérateur «*» qui indique au compilateur *C* qu'il doit lire la valeur adressée par le pointeur. Pour lire la valeur du premier pixel de la mémoire écran, il faut procéder de la manière suivante :

```
byte *ptr;
byte pixel;

ptr = (byte *)MK_FP(0xA000,0x0000);
pixel = *ptr;
```

Pour accéder au second pixel, il suffit d'incrémenter le pointeur avec l'opérateur ++.

```
byte *ptr;
byte pixel1;
byte pixel2;

ptr = (byte *)MK_FP(0xA000,0x0000);
pixel1 = *ptr;
ptr++;
pixel2 = *ptr;
```

Pour accéder à un pixel particulier de l'écran, il faut calculer sa position, c'est-à-dire son offset. Il n'y a pas de calcul à faire pour le segment, puisque toute la mémoire écran est située dans le segment 0xA000. Nous avons déjà vu que l'écran est organisé en 200 lignes de 320 pixels. La formule de calcul pour l'offset d'un pixel est la suivante :

```
offset = py*320 + px;
```

Voici un listing qui permet de connaître la couleur d'un pixel quelconque :

```
byte *ptr;
byte pixel;
int  px, py;
unsigned int offset;

// CALCUL POSITION
px = 100;
py = 100;
offset = py*320 + px;

// LECTURE PIXEL
ptr = (byte *)MK_FP(0xA000,offset);
pixel = *ptr;
```

Ecriture dans la mémoire vidéo

Il est aussi facile d'écrire dans la mémoire vidéo que d'y lire. Les pointeurs sont vraiment des outils précieux ! Voici un petit programme qui écrit la valeur 125 dans le premier pixel de l'écran :

```
byte *ptr;

ptr = (byte *)MK_FP(0xA000,0x0000);
*ptr = 125;
```

La routine suivante remplit les 64 000 pixels de l'écran avec la même couleur. Il initialise le pointeur ptr avec l'adresse de début de la mémoire vidéo, puis utilise une boucle pour balayer les 64 000 octets de l'écran. Vous vous demandez peut-être pourquoi la variable n est du type *unsigned int* et *non int* ? Les variables de type *int* sont définies de la valeur -32768 à la valeur +32767. Cela ne va pas dans le cadre de notre programme où une boucle doit s'exécuter 64 000 fois ! C'est pourquoi l'indice de boucle est du type *unsigned int* dont la valeur peut varier entre 0 et 65 535. Si la boucle doit s'exécuter plus de 65 535 fois, il faut utiliser une variable de type *Long* qui peut contenir des nombres entiers de grande valeur. La valeur 64 000 est suivie par la lettre «*u*» qui signifie *unsigned*.

```
void remplissage_ecran(char c)
{
byte*      ptr;
unsigned int n;

 ptr=(byte *)MK_FP(0xA000,0x000);
 for (n=0; n<64000u; n++)
 {
   *ptr = c;
   ptr++;
 }
}
```

Pour finir

La manipulation directe de la mémoire écran offre de nombreuses possibilités pour la réalisation de programmes graphiques performants. Nous allons beaucoup l'utiliser dans les articles à venir, pour afficher des sprites, réaliser des effets spéciaux graphiques, étudier les techniques de scrollings, etc.

Patrick Leclercq

Les compilateurs C et le DOS

Les nouveaux compilateurs 32 bits, comme le Visual C++ 4.0, sont conçus uniquement pour générer du code 32 bits pour Windows. Si vous voulez écrire des programmes DOS, il faut utiliser des compilateurs plus anciens, comme le Visual C++ 1.5 ou le Borland C 4.0.

Chargement d'une image BMP

LE FORMAT BMP EST LE PLUS SIMPLE ET LE PLUS RÉPANDU DES FORMATS GRAPHIQUES. LA GÉNÉRALISATION DE WINDOWS EN A FAIT UN STANDARD INCONTOURNABLE. C'EST POURQUOI CET ARTICLE VOUS EXPLIQUE COMMENT CHARGER LES IMAGES BMP DANS VOS PROGRAMMES.

▲ *Voici un exemple des images BMP que vous trouverez sur le disque.*

Les fichiers graphiques ne contiennent pas que des images. Ils renferment aussi des informations sur la taille des images, la manière dont elles sont stockées et la palette de couleurs utilisée. C'est pourquoi un fichier *BMP* est constitué de 4 éléments différents. Le premier élément est une structure de 14 octets qui porte le nom de *BITMAPFILEHEADER*. Sa structure est la suivante :

```
Un octet contenant la lettre «B»
Un octet contenant la lettre «M»
Un DWORD (32 bits non signé) contenant la taille de l'image
Un WORD (16 bits non signé) réservé par le constructeur
Un WORD réservé par le constructeur
Un DWORD contenant la position de l'image binaire à partir du début du fichier.
```

La structure suivante porte le nom de *BITMAPINFOHEADER*. Sa taille n'est pas fixe. Le constructeur s'est réservé le droit d'étendre cette zone pour y stocker de nouvelles informations, dans un futur plus ou moins proche. Sa structure est la suivante :

```
Un DWORD : Taille de la structure
Un LONG : Hauteur de l'image
Un LONG : Largeur de l'image
Un WORD : Nombre de plans de pixels - toujours 1
Un WORD : Nombre de bits par plan (1, 4, 8, 16, 24 ou 32)
Un DWORD : Mode de compression (0=non compression)
Un DWORD : Taille de l'image en octets (0=image non-compressée)
Un LONG : Résolution en pixel/mètre x
Un LONG : Résolution en pixel/mètre y
Un DWORD : Nombre d'index de couleurs utilisées dans l'image
Un DWORD : Nombre de couleurs vraiment importantes dans l'image.
```

Cette structure contient beaucoup d'informations pour une simple image. On y apprend, entre autres, que le format *BMP* peut contenir des données graphiques compressées, ce qui risque de compliquer l'écriture d'une routine de chargement. Heureusement pour nous, les images *BMP* ne sont pratiquement jamais compressées !

La structure suivante est un tableau qui contient la palette de couleurs de l'image. Les couleurs sont stockées sous la forme de nuance *RVB* (Rouge, Vert, Bleu). Le codage de chaque index de couleur occupe 4 octets. Le premier octet est vide, tandis que les octets 2, 3 et 4 contiennent les composantes Bleu, Vert et Rouge. Le stockage de la palette d'une image 256 couleurs nécessite 1 024 octets (256*4).

```
1er octet : rien
2e octet : composante Bleu couleur 0
3e octet : composante Vert couleur 0
4e octet : composante Rouge couleur 0
5e octet : rien
6e octet : Composante Bleu couleur 1
7e octet : composante Vert couleur 1
8e octet : composante Rouge couleur 1
9e octet : ....
```

Et enfin, la dernière structure est l'image elle-même ! Elle est stockée à l'envers, c'est-à-dire que la première ligne du fichier est la dernière ligne de l'écran. Il faut tenir compte de ce décalage lors du chargement de l'image.

Chargement d'une image BMP

La structure d'une image *BMP* est assez complexe, même si les autres formats graphiques sont encore plus complexes ! Cela tient au fait que les concepteurs du *BMP* ont voulu créer un format graphique universel, utilisable quelle que soit la résolution et le nombre de couleurs de l'image. Bref, il est complexe d'écrire une routine de chargement *BMP* fonctionnant dans tous les cas de figure possibles. Heureusement, ce n'est pas notre intention ! Le but de cet article est d'obtenir une routine de chargement d'images *BMP* qui fonctionne uniquement dans le mode *VGA* 320x200 en 256 couleurs. C'est le mode graphique le plus utilisé pour les applications *DOS*.

▲ *Voici une belle image BMP 3200x200 en 256 couleurs.*

Une image *BMP* 320x200 en 256 couleurs non compressée mesure 65 078 octets. Le premier bloc d'informations est placé au début du programme. Le second bloc d'informations commence à partir du 15e octet du fichier. Le codage de la palette de couleurs commence à partir du 54e octet et occupe 1 024 octets. L'image proprement dite est stockée à partir de 1 079 octets du fichier. Elle occupe 64 000 octets (200 lignes de 320 octets=64 000). Afin de simplifier la programmation, nous allons considérer que toutes les images *BMP* 320x200 ont cette structure.

Les étapes de chargement

La première étape du chargement est l'ouverture d'un fichier binaire. Ce type de fichier est très précieux pour les applications graphiques car il permet d'accéder à tous les octets d'un fichier. Cela se fait avec l'instruction *fopen()*. La chaîne «*rb*» indique au compilateur *C* que le fichier doit être ouvert en lecture (*read*) et qu'il est du type binaire (*binary*).

```
FILE *fic

fic = fopen(nom_image, «rb»);
```

A partir du moment où le fichier contient une image *BMP* 320x200 en 256 couleurs, nous n'avons pas besoin de consulter l'en-tête pour connaître les caractéristiques de l'image. C'est pourquoi nous pouvons passer directement à la palette de couleurs. La fonction *fseek()* permet de positionner le fichier à une position quelconque. La palette de couleurs commence au 54e octet du fichier. Mais comme la numérotation des octets commence à la position 0, le programme doit se placer sur l'octet 53 pour accéder à la palette de couleurs.

```
fseek(fic, 53, SEEK_SET);
for (c=0; c<256; c++)
{
 rien=(64*getc(fic))/256;
 bleu=(64*getc(fic))/256;
 vert=(64*getc(fic))/256;
 rouge=(64*getc(fic))/256;
 pal[c][0]=rouge;
 pal[c][1]=vert;
 pal[c][2]=bleu;
}
```

Nous avons vu que chaque couleur est constituée d'un mélange de trois couleurs élémentaires : le Bleu, le Vert et le Rouge. C'est ce qu'on appelle le codage *RVB*. Les composantes *RVB* des fichiers *BMP* sont codées sur 256 valeurs différentes. Cela veut dire que chaque couleur élémentaire peut avoir 256 teintes différentes. En mélangeant ces teintes élémentaires, on peut obtenir plus de 16 millions de couleurs. C'est énorme ! Cependant, cela pose un problème car les cartes *VGA* ne sont pas conçues pour supporter autant de couleurs. La norme *VGA* de base prévoit que les composantes *RVB* ne peuvent prendre que 64 valeurs différentes et non 256. Cela ne permet d'avoir que 162 144 couleurs différentes. C'est pourquoi notre routine de chargement multiplie les composantes *RVB* par 64 et les divise par 256. Cela convertit le codage *RVB* 16 millions de couleurs en codage *RVB* 162 144 couleurs. La plupart des cartes graphiques actuelles ont des modes graphiques avec 16 millions de couleurs. Mais cela n'empêche pas leurs modes graphiques 256 couleurs d'être toujours limités à seulement 162 144 couleurs !

Au fur et à mesure de la lecture du fichier, les composantes *RVB* sont stockées dans un tableau d'octets. Une fois les 256 couleurs en mémoire, le programme appelle la routine *setpalette()* en lui passant l'adresse du tableau. Cette routine remplace automatiquement la palette de couleurs de l'écran par la palette de couleurs de l'image. L'opération est très rapide.

```
setpalette(&pal[0][0]);
```

Et Windows ?

Cet article traite uniquement du chargement d'une image BMP 320x200 en 256 couleurs dans un programme DOS. Nous n'avons pas parlé du chargement des images BMP sous Windows car la plupart des langages de programmation Windows possèdent leurs propres routines de chargement BMP. Néanmoins, nous aborderons prochainement ce sujet, en expliquant comment charger une image BMP dans un programme Windows écrit avec le SDK Game.

Cette routine *setpalette()* n'est pas une fonction standard du *C*. Elle a été écrite à partir d'une routine graphique de la carte *VGA*. Celle-ci possède un certain nombre de fonctions qui peuvent être appelées à partir d'une interruption système. Nous avons déjà parlé de l'utilisation des interruptions dans le dernier numéro de PC FUTE à propos de la gestion de la souris en *Qbasic*. Nous reparlerons des interruptions et du langage *C* dans un prochain article. Le seul paramètre dont *setpalette()* ait besoin est l'adresse d'un tableau d'octets de 768 octets (3x256 octets) qui contient le codage *RVB* de toutes les 256 couleurs de la nouvelle palette. Voici son listing :

```
void setpalette(byte *pal)
{
 union  REGS  reg;
 struct SREGS regs;
 reg.x.ax = 0x1012;
 regs.es=_FP_SEG(pal);
 reg.x.dx=_FP_OFF(pal);
 reg.x.bx=0x0;
 reg.x.cx=256;
 int86x(0x10,&reg,&reg,&regs);
}
```

Une fois la palette de couleurs modifiée, la routine peut charger les 200 lignes de l'image. La première ligne de l'image commence à l'octet 1 078 du fichier. C'est pourquoi le programme doit encore utiliser la fonction *fseek()* pour positionner le fichier sur cet octet. Les lignes sont stockées à l'envers. Le programme doit donc commencer la lecture par la ligne 199 et finir par la ligne 0. Il y a 200 lignes, mais la numérotation se fait à partir de la ligne 0 et c'est pourquoi la dernière ligne porte le numéro 199 et non 200.

```
fseek(fic, 1078, SEEK_SET);
ligne = 199;
for (n=0; n<200; n++)
{
 // CALCUL ADRESSE LIGNE
 ptr = (byte *)MK_FP(0xA000,
                     ligne*320);
 // CHARGEMENT LIGNE
 fread (ptr, 320, 1, fic);
 ligne = ligne-1;
}
```

Le chargement de chaque ligne en mémoire se fait à l'aide de la fonction *fread()* qui permet de lire un bloc d'octets et de la transférer directement à une adresse quelconque de la mémoire. C'est la technique la plus rapide qui existe en *C* pour lire des données disques. Le programme utilise l'instruction *MK_FP()* pour calculer l'adresse des lignes dans la mémoire écran. Vous trouverez dans ce magazine un article qui explique la structure de la mémoire vidéo du mode graphique *VGA*.

▲ *Ce tableau ancien est une image BMP 320x200.*

La bibliothèque GRAPHVGA

Afin de simplifier l'utilisation de la routine *loadBMP(,)* nous avons écrit une nouvelle bibliothèque graphique. Celle-ci regroupe toutes les fonctions graphiques présentées dans les deux derniers numéros de PC FUTE. Elle permet de charger une image *BMP* de manière très simple, comme le montre l'exemple suivant :

```
#include <stdio.h>
#include <conio.h>
#include «graphvga.h»

void main (void)
{
 init_vga256();
 getch();
 loadBMP(«chats.bmp»);
 getch();
 close_vga256();
}
```

Vous trouverez sur le disque une version de la bibliothèque *GRAPHVGA.H* pour le *Visual C* et une autre version pour le *Borland C*. L'adaptation à un autre compilateur *C* est facile à faire.

Pour finir

Vous pouvez utiliser la routine *loadBMP()* pour afficher des images *BMP* sur l'écran. Mais son principal intérêt réside dans la possibilité de récupérer des graphismes pour les programmes graphiques et les jeux. Vous trouverez un exemple de cette technique dans un autre article de ce magazine, où l'auteur explique comment générer la carte d'un vaste pays, à partir de petits éléments graphiques de base, dessinés par un graphiste avec un logiciel de dessin.

Patrick Leclercq

VOUS AIMEZ CIVILISATION ? VOUS AVEZ DES IDÉES SUR LA MANIÈRE DONT DOIT FONCTIONNER LE JEU DE SIMULATION IDÉAL ? DANS CE CAS, POURQUOI NE PAS CRÉER VOTRE PROPRE LOGICIEL ? BIENVENUE DANS LE MONDE DES CRÉATEURS D'UNIVERS !

Programmation d'un jeu de stratégie

▲ *Voici un écran de Civilisation montrant un affrontement entre l'empire romain et les Zoulous. Ceux-ci ont violé un traité de paix et attaqué les forces de Rome par surprise. Qui va gagner ?*

Depuis quelques années, les jeux de simulation stratégique, comme *Colonisation, Populus, Masters of Magic, Orion, César* et *Ufo* ont beaucoup de succès. C'est surtout les plus de 25 ans qui aiment ce type de jeux. Les plus jeunes préfèrent les jeux d'action et les jeux d'aventure. Le jeu le plus connu est sans conteste *Civilisation* qui, depuis sa sortie en 1992, connaît un énorme succès. Malgré son intérêt ludique, ce jeu n'a rien de fabuleux sur le plan technique. C'est l'idée de base et la manière dont elle a été traitée qui sont importantes. On peut programmer d'excellents jeux sans être un virtuose de l'informatique. C'est pourquoi nous avons décidé d'écrire quelques articles dévoilant les secrets de la programmation d'un jeu de type *Civilisation.*

Qu'est-ce que Civilisation ?

C'est un jeu où le joueur doit gérer l'évolution d'une civilisation. Il commence en l'an -4000 avec un petit village et doit créer une grande civilisation. Pour cela, il faut fonder des villes, explorer le monde, développer le commerce et la recherche scientifique, sans oublier de créer des unités militaires et d'aller casser la figure au voisin ! Le jeu se déroule sur la carte d'un monde, découpé en lignes et en colonnes. Cela ressemble beaucoup à un jeu de société, ce qui est normal, quand on sait que *Civilisation* a été beaucoup inspiré par le jeu de plateau éducatif *Civilisation* qui retrace l'évolution de peuples primitifs jusqu'à l'âge de fer.

Le jeu commence avec une unité de colons qui permet de créer une ville. Ensuite, cette ville grandit et crée des unités militaires pour explorer le monde et des colons pour exploiter la terre et créer d'autres villes. Les ressources produites par les villes peuvent également servir à construire de nouveaux bâtiments. Ils apportent des avantages et de nouvelles possibilités, accélérant notamment l'évolution technologique qui est le coeur du jeu. La vitesse d'acquisition des technologies dépend des ressources consacrées à la recherche et à l'érudition. Mais les autres peuples sont agressifs et s'opposent à la progression de l'empire, en s'emparant des villes et en volant vos technologies. Il faut donc répartir intelligemment les ressources entre la fabrication d'unités militaires et les projets civils, tout en sachant que l'acquisition de nouvelles technologies permet de créer des unités militaires plus performantes.

Les problèmes techniques

La programmation d'un jeu de type *Civilisation* pose un certain nombre de problèmes intéressants. Il faut gérer l'affichage de la carte, s'occuper

Système : DOS
Langage : Visual C et Borland C

du déplacement des unités sur le terrain, traiter les combats entre unités, gérer les villes et les constructions civiles, faire évoluer la technologie en fonction de la recherche et surtout, faire réagir intelligemment les civilisations ennemies, de manière à offrir un vrai défi au joueur. Cela représente beaucoup de travail. Ce mois-ci, nous allons étudier l'affichage de la carte.

Gestion de la carte

Comme vous pouvez le voir sur les images illustrant l'article, la carte de *Civilisation* est dessinée à partir d'éléments graphiques rectangulaires. Chaque élément mesure 16x16 pixels. La carte totale est constituée de 80x50 carrés. L'écran étant trop petit pour contenir la carte dans sa totalité, le programme se contente d'afficher une fenêtre de 15x12 élément. Cela occupe à peu près les 3/4 de l'écran. Le reste de l'écran est utilisé pour afficher les informations du jeu.

La technique la plus simple pour stocker la carte en mémoire est d'utiliser un tableau à deux dimensions.

Et windows ?

La plupart des programmes DOS travaillent en mode VGA 320x200, c'est pourquoi les éléments graphiques de base ont une taille de 16x16 pixels. En revanche, avec Windows et les jeux récents, la norme graphique pour les jeux est le 640x480. Cela permet de dessiner de superbes éléments graphiques de 32x32 pixels.

Chaque élément de la carte est un *byte*, c'est-à-dire un entier non signé dont la valeur peut varier entre 0 et 255. Il peut donc y avoir 256 éléments de terrain différents. C'est énorme. En fait, *Civilisation* n'utilise pas plus d'une centaine d'éléments pour dessiner toutes les cartes.

```
#define MAX_COLONNES  80
#define MAX_LIGNES    50

byte carte[MAX_LIGNES]
          [MAX_COLONNES];
```

Stockage des éléments graphiques

Le logiciel doit conserver quelque part en mémoire les éléments graphiques servant à dessiner la carte. La méthode la plus simple pour réaliser cela est d'utiliser une bibliothèque d'images. Les routines que nous allons écrire sont conçues pour le *Visual C*. Mais l'adaptation à un autre compilateur *C* ne pose pas de problème. Il s'agit juste de changer le nom de quelques fonctions graphiques. La disquette contient une version *Borland C* de la bibliothèque graphique.

Une bibliothèque d'images est un outil qui permet de stocker des graphismes en mémoire et de les afficher à la demande. Pour la créer, nous avons besoin de routines capables de manipuler des blocs de pixels. La fonction *_getimage()* du *Visual C* permet de stocker une image en mémoire. D'après l'aide intégrée du *Visual C*, sa syntaxe est la suivante :

```
void _far _getimage(
          short x1, short y1,
          short x2, short y2,
          char __huge *image );
```

C'est un peu complexe, à cause des pointeurs 32 bits qui permettent d'agir sur la totalité de la mémoire, quel que soit le mode d'adressage mémoire du compilateur *C*. A condition d'utiliser le mode mémoire «*Large*», on peut considérer que la syntaxe est la suivante :

```
void getimage(short x1, short y1,
              short x2, short y2,
              char *image);
```

La position et la taille du bloc de pixels sont définies par les points (*x1,y1*) et (*x2,y2*). La variable image est un pointeur de type *char* qui contient l'adresse de la zone mémoire où doit être stockée l'image.

Il faut réserver une zone de mémoire pour stocker une image. Pour cela, il faut que le programme connaisse à l'avance le nombre d'octets nécessaires au stockage de l'image. C'est la tâche de la fonction *_imagesize()*, dont voici la syntaxe :

```
long _sizeimage(short x1, short y1,
                short x2, short y2);
```

Elle retourne un entier de type long qui indique le nombre d'octets nécessaires à la mémorisation de l'image définie par (*x1,y1*)-(*x2,y2*). Il ne reste plus qu'à utiliser la fonction *malloc()* pour allouer la mémoire nécessaire. Voici un programme d'exemple qui met cela en pratique :

```
int  x1, y1, x2, y2;
long taille;
char *adr;

// CALCUL TAILLE IMAGE
taille = _imagesize(x1,y1,x2,y2);
// RESERVATION MEMOIRE
adr=(char *)malloc((size_t)taille);
// STOCKAGE IMAGE EN MEMOIRE
_getimage(px,py,px2,py2, adr);
```

Les positions (*x1,y1*) et (*x2,y2*) sont stockées dans des variables de type int. Cela ne pose pas de problème au compilateur car ces deux types sont compatibles. Ce n'est le cas avec la fonction *_imagesize()* qui retourne une valeur de type *long*, alors que la fonction *malloc()* a besoin d'une valeur de type *size_t*. Pour éviter un message d'avertissement, il faut forcer la conversion de la valeur long en type *size_t*. Cela se fait en ajoutant la déclaration (*size_t*) devant la variable de type *long*.

```
// CONVERSION long en size_t
malloc((size_t)taille);
```

De la même manière, *malloc()* retourne un pointeur indifférencié (*void **), alors que la fonction *_getimage()* a besoin d'un pointeur sur un caractère (*char **). C'est pourquoi le programme convertit le pointeur fourni par *malloc()* en un pointeur de type caractère.

```
adr=(char *)malloc((size_t)taille);
```

Il faut procéder à ces conversions de types, afin que le compilateur *C* ne génère pas un message d'avertissement (*warning*) chaque fois que les types de données ne sont pas strictement identiques. Ce contrôle strict des données agace fréquemment les débutants en *C*. C'est néanmoins nécessaire car cela permet de détecter certaines erreurs de programmation. Le pointeur indifférencié (*void **) est compatible avec tous les autres types de pointeurs, c'est-à-dire qu'il n'y a pas besoin de faire une conversion pour affecter un pointeur (*void **) à un pointeur (*char **). Le listing suivant ne génère donc pas de message d'avertissement :

```
adr = malloc((size_t)taille);
```

Stockage d'une liste d'images

Nous savons maintenant comment stocker une image en mémoire. Il nous reste à étudier comment stocker plusieurs images dans la mémoire. La technique la plus simple est d'utiliser un tableau pour stocker les adresses des images. Cela se fait de la manière suivante :

```
#define MAX_IMAGES 100

char *_images[MAX_IMAGES];
```

La routine *saisie_image()* permet de stocker une image dans le tableau. Les paramètres de saisie sont la position (*px,py*) de l'image, sa taille (*tx,ty*) et son numéro. A partir de ces données, la routine réserve la place nécessaire au stockage de l'image, la transfère dans la mémoire et mémorise l'adresse de stockage dans le tableau *_images[]*.

```
void saisie_image(int px, int py,
                  int tx, int ty,
                  int n)
{
 int  px2, py2;
 long taille;
 char *adrimage;

 px2 = px+tx-1;
 py2 = py+ty-1;

 taille =_imagesize(px, py, px2, py2);
 adr =(char *) malloc((size_t)taille);
 _getimage(px,py,px2,py2, adr);
 _images[n]=adrimage;
}
```

Pour afficher une image sur l'écran, il faut utiliser la routine *affichage_image()*. Elle est écrite à partir de la fonction *_putimage()* qui permet d'afficher sur l'écran une image préalablement sauvée par *_getimage()*. La routine commence par lire l'adresse de l'image dans le tableau *_images[]*, puis elle utilise *_putimage()* pour l'affichage.

```
void affichage_image(int px,
                     int py,
                     int n)
{
 char *adr;

 adr = _images[n];
 if (adr!=NULL)
  _putimage(px, py, adr, _GPSET);
}
```

La routine utilise un petit système de sécurité pour éviter qu'il se produise une catastrophe lorsque le programme tente d'afficher une image qui n'existe pas. L'affichage ne se fait que si l'adresse de l'image est différente de la valeur *NULL*. Cela ne fonctionne que si toutes les cases du tableau *_images[]* contiennent la valeur *NULL* au début du programme. C'est la tâche de la fonction *init_images()* qui doit impérativement être exécutée au début du programme.

```
void init_images (void)
{
 int n;

 for (n=0; n<MAX_IMAGES; n++)
  _images[n]=NULL;
}
```

La mémoire réservée par *malloc()* doit être libérée à la fin du programme. Sans cela, la mémoire reste allouée et ne peut être utilisée par un autre programme. C'est la tâche de la fonction *free()*. Il suffit de lui passer l'adresse d'un bloc de mémoire pour que ce bloc soit libéré par le système. La routine *effacer_images()* permet de libérer automatiquement toutes les zones mémoires utilisées pour le stockage des images du logiciel. Elle doit impérativement être exécutée à la fin du programme.

```
void effacer_images (void)
{
 int n;
 for (n=0; n<MAX_IMAGES; n++)
  if (_images[n]!=NULL)
    free (_images[n]);
}
```

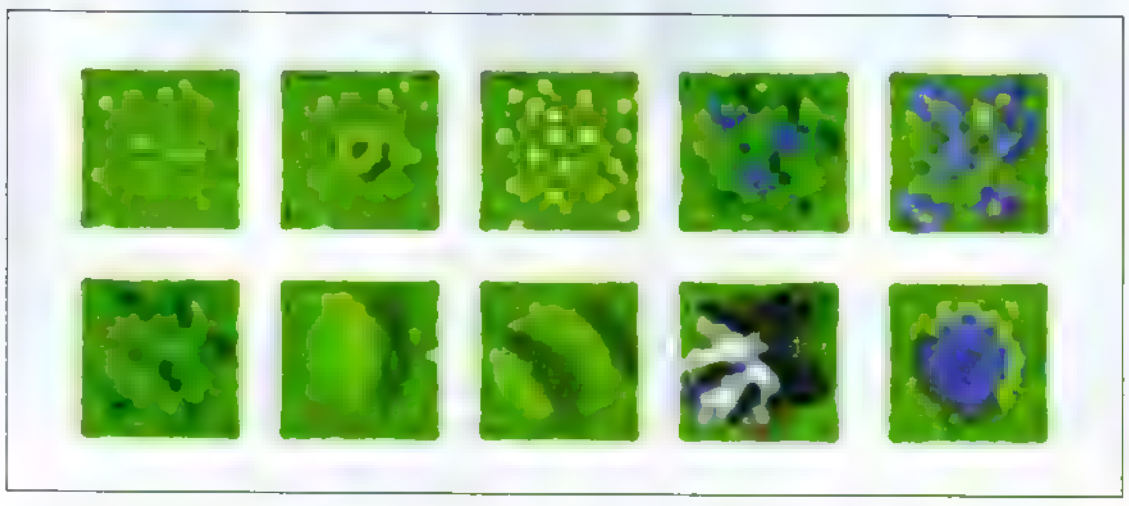
Création des éléments graphiques

Comme nous l'avons vu plus haut, l'image de la carte est constituée de petits éléments rectangulaires de 16x16 pixels. Chaque image représente un élément du terrain : plaine, forêt, collines, montagnes, rivière, lac, etc. Ils sont généralement dessinés par un graphiste. C'est une tâche délicate car les éléments doivent s'associer de manière précise, sans donner l'impression d'un damier. Afin d'illustrer cet article, nous avons «collé» un graphiste devant une image de *Civilisation*, avec pour mission de dessiner des éléments ressemblants. Il nous a «pondu» 10 éléments de base : 5 éléments de plaines, une forêt, deux collines, une montagne et un lac. Cela suffit pour réaliser notre programme d'exemple.

Les éléments graphiques sont stockés dans l'image *elements.bmp*. Pour les récupérer, il faut charger l'image sur l'écran. C'est facile à réaliser avec la fonction *loadBMP()* dont le fonctionnement est détaillé dans un autre article de ce magazine. La routine *chargement_graphismes()* s'occupe de charger les différents éléments en mémoire. Elle commence par charger l'image *elements.bmp* sur l'écran, puis stocke chaque élément en mémoire, à l'aide de la fonction *saisie_image()*.

```
void chargement_graphismes()
{
 int tx, ty;

 tx = 16;
 ty = 16;

 loadBMP(«elements.bmp»);

 saisie_image(6,9,tx,ty, PLAINE1);
 saisie_image(26,9,tx,ty,PLAINE2);
 saisie_image(46,9,tx,ty,PLAINE3);
 saisie_image(66,9,tx,ty,PLAINE4);
 saisie_image(87,9,tx,ty,PLAINE5);
 saisie_image(6,30,tx,ty,FORET);
 saisie_image(26,30,tx,ty,COLLINE1);
 saisie_image(46,30,tx,ty,COLLINE2);
 saisie_image(66,30,tx,ty,MONTAGNE);
 saisie_image(87,30,tx,ty,LAC);
}
```

La plupart des logiciels de dessin affichent la position courante de la souris. Cela nous a permis de déterminer la position de chaque élément sur l'écran. Nous avons utilisé le logiciel *Paint* fourni gratuitement avec *Windows 95*.



▲ *Voici les 10 éléments graphiques qui vont nous servir à créer nos propres cartes.*

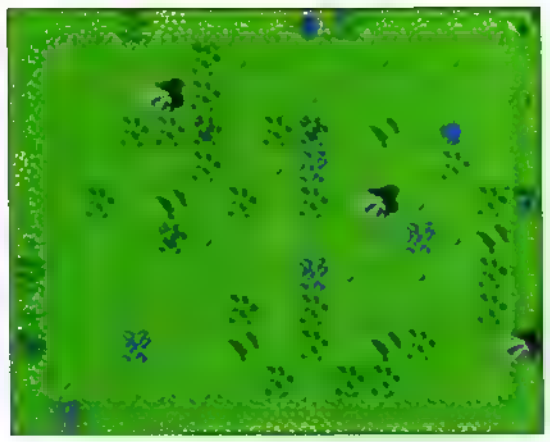
Pour faciliter la manipulation des éléments dans le programme, nous leur avons associé des constantes symboliques. C'est plus facile à utiliser que des nombres.

```
#define PLAINE1   0
#define PLAINE2   1
#define PLAINE3   2
#define PLAINE4   3
#define PLAINE5   4
#define FORET     5
#define COLLINE1  6
#define COLLINE2  7
#define MONTAGNE  8
#define LAC       9
```

Génération aléatoire de la carte

La carte peut être dessinée avec un éditeur ou générée aléatoirement. Le logiciel *Civilisation* utilise les deux méthodes. Il est possible de jouer sur un monde identique à la Terre ou sur des mondes générés aléatoirement. La Terre a été créée par les programmeurs de *Civilisation* avec un éditeur de carte et les mondes aléatoires sont fabriqués par des routines spécialisées. C'est cette technique que nous avons décidé de reproduire ici.

La méthode la plus simple pour fabriquer une carte est de faire une boucle et de faire un tirage aléatoire pour chaque case, avec une probabilité équivalente pour chaque élément. Ce n'est une bonne méthode car cela produit des cartes trop régulières. Il faut utiliser une méthode différente. C'est la routine *generation_carte()* qui fabrique la carte utilisée par notre programme d'exemple. Elle balaie l'ensemble de la carte et tire aléatoirement un élément à chaque case. Tous les éléments n'ont pas la même probabilité d'apparition. Par exemple, possibilité d'avoir une forêt 10%, une colline ▶



▲ *Voici un exemple de carte aléatoire générée par le programme AFFCARTE.C.*

5%, une montagne 3% et un lac 2% et si aucun de ces éléments n'est sélectionné, ce sera une plaine, ce qui équivaut à une probabilité de 80%.

```
void generation_carte()
{
 int l,c;

 for (l=0; l<MAX_LIGNES; l++)
  for (c=0; c<MAX_COLONNES; c++)
  {
   if (aleatoire(10))
     selection_foret(l, c);
   else if (aleatoire(5))
     selection_colline(l, c);
   else if (aleatoire(2))
     selection_montagne(l, c);
   else if (aleatoire(2))
     selection_lac(l, c);
   else
     selection_plaine(l,c);
  }
}
```

C'est la fonction *aleatoire()* qui permet de déterminer si une probabilité se réalise ou non. Elle effectue un tirage aléatoire compris entre 1 et 100 et compare le résultat avec le paramètre de la fonction. La probabilité se réalise si elle est inférieure au tirage. Prenons par exemple une probabilité de 10%. Elle se réalise si le tirage est compris entre 1 et 10.

```
int aleatoire(int n)
{
 int tirage;
 tirage = rnd(100)+1;
 if (tirage<=n) return (1);
          else return (0);
}
```

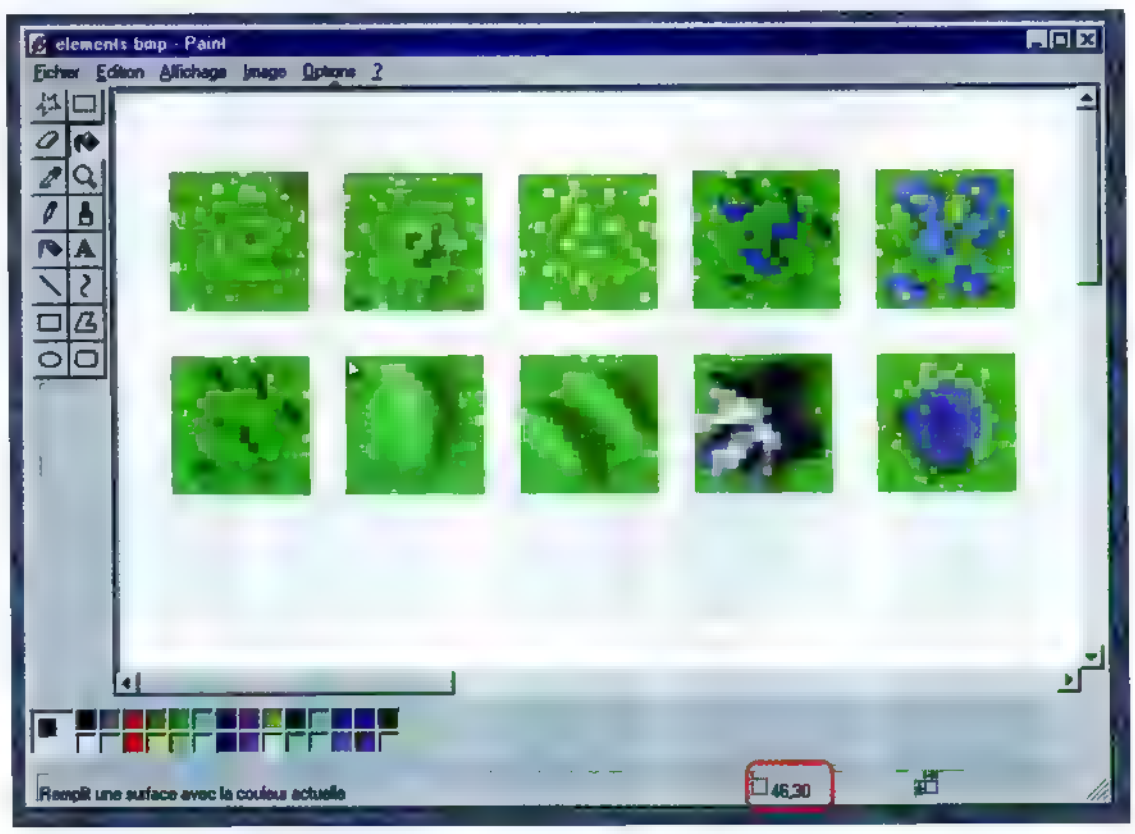
L'ensemble des probabilités constitue la courbe de répartition des objets sur la carte. En changeant ces probabilités, vous pouvez modifier l'allure de la carte. Cela permet de découper la carte en régions et d'utiliser des probabilités différentes pour chaque région. La courbe de répartition d'une zone montagneuse est différente de celle d'un désert ou d'une plaine.

```
void selection_plaine(ligne,
                      colonne)
{
 int element;

 if (aleatoire(10))
   element=PLAINE2;
 else if (aleatoire(10))
   element=PLAINE3;
 else if (aleatoire(5))
   element=PLAINE4;
 else if (aleatoire(5))
   element=PLAINE5;
 else
   element=PLAINE1;

 carte[ligne][colonne]=element;
}
```

Il est possible de faire un second tirage aléatoire pour désigner quelle est l'image qui doit correspondre au tirage. Prenons par exemple le cas de la plaine. Il y a 5 éléments différents pour représenter la même chose. En fait, il y a un dessin principal et 4 éléments pour éviter la monotonie. Les éléments *PLAINE2* et *PLAINE3* ont une probabilité d'apparition de 10%, tandis que les éléments *PLAINE4* et *PLAINE5* ont une probabilité de 5%. Si aucun de ces éléments n'est tiré, c'est l'élément *PLAINE1* qui est sélectionné. Sa probabilité d'apparition est donc de 70% (100-10-10-5-5=70).

Affichage de la carte

Comme nous l'avons vu plus haut, la carte est trop grande pour être affi-



▲ *Pour connaître la position exacte d'un élément graphique sur une image, il faut charger l'image dans un logiciel de dessin, activer le mode loupe et placer la souris dans le coin haut gauche de l'élément.*

chée en totalité. Nous ne pouvons en afficher qu'une partie sur l'écran. La fenêtre de visualisation de *Civilisation* étant de 15x12, nous allons refaire la même chose. La première chose à faire est de définir la taille de la fenêtre de visualisation avec des constantes symboliques.

```
#define LIGNES_FENETRE     12
#define COLONNES_FENETRE   15
```

Nous avons aussi besoin de savoir à quel endroit de l'écran se fait l'affichage de la carte. C'est le rôle des constantes *PX_CARTE* et *PY_CARTE*.

```
#define PX_CARTE 40
#define PY_CARTE  4
```

Et surtout, le programme a besoin de savoir quelle est la portion de carte à afficher. C'est la tâche des variables globales *ligne_carte* et *colonne_carte* qui conservent en permanence la position de la fenêtre d'affichage sur la carte.

```
int ligne_carte;
int colonne_carte;
```

L'affichage de la carte sur l'écran se fait avec la fonction *affichage_fenetre()*. Comme vous pouvez le voir, cette routine n'est pas très compliquée ! Elle utilise deux boucles pour balayer toutes les cases visibles. Pour chaque case, elle consulte le tableau *carte[]* pour connaître l'élément correspondant, calcule la position d'affichage et dessine l'élément graphique sur l'écran. Pour afficher la carte à une nouvelle position, il faut changer la valeur des variables globales *ligne_carte* et *colonne_carte* et appeler à nouveau la routine *affichage_fenetre()*.

```
void affichage_fenetre()
{
 int l, c;
 int ligne;
 int colonne;  int px, py;
 int n;

 ligne=ligne_carte;
 for (l=0; l<LIGNES_FENETRE; l++)
 {
  colonne=colonne_carte;
  for (c=0; c<COLONNES_FENETRE; c++)
  {
   n=carte[ligne][colonne];
   px = PX_CARTE + c*16;
   py = PY_CARTE + l*16;
   affichage_image(px, py, n);
   colonne++;
  }
  ligne++;
 }
}
```

▲ *Master Of Magic est un jeu ressemblant beaucoup à Civilisation. La principale différence est le remplacement de la technologie par la magie.*

L'affichage de la carte est plus ou moins rapide selon la puissance de votre machine et le type de carte graphique. Il est possible de l'améliorer en créant de nouvelles fonctions graphiques écrivant directement dans la mémoire écran.

Vous trouverez sur le disque le programme *AFFCARTE.C* qui utilise toutes ces techniques pour générer une carte aléatoire et l'afficher sur l'écran. Il est possible de scroller sur la carte en utilisant les flèches du clavier. Pour quitter la démonstration, il faut presser sur la touche [*Echap*]. Le listing est constitué de deux fichiers sources. Le premier contient le programme proprement dit et le second contient les routines graphiques nécessaires à son fonctionnement. Cela permet de faciliter l'adaptation du logiciel à un autre compilateur *C*. La disquette contient une version *Visual C* du programme et une autre version en *Borland C*.

Pour finir

Vous savez maintenant comment fonctionne l'affichage d'un jeu de simulation. Vous pouvez l'étudier et l'améliorer. La prochaine fois, nous verrons le déplacement des unités et le déroulement d'un combat. C'est un élément important des jeux à orientation stratégique.

Patrick Leclercq

Une idée pour les graphistes

Vous voulez participer à notre concours sur la création d'images vidéo ? Dans ce cas, pourquoi ne pas dessiner un jeu complet d'éléments permettant de réaliser de superbes cartes type Civilisation ?

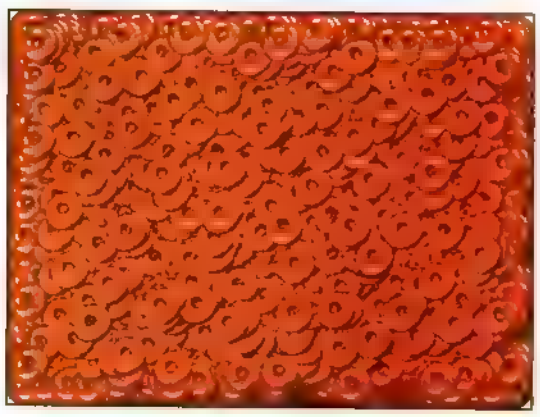
Utilisation de DirectDraw

Le système DirectX est en train de révolutionner la manière d'écrire des logiciels graphiques pour Windows 95. Il se compose de plusieurs modules, dont DirectDraw. La tâche de celui-ci est d'accélérer les opérations graphiques 2D. Nous allons voir comment utiliser concrètement DirectDraw pour réaliser des opérations graphiques ultra rapides. Bienvenue dans le monde du futur !

DirectX est une extension graphique de *Windows 95*, conçue pour faciliter la réalisation de jeux et de programmes graphiques performants. Un logiciel écrit avec le système *DirectX* est capable de court-circuiter le système graphique de *Windows 95* et d'accéder directement à la carte vidéo.
Pour écrire des programmes exploitant *DirectX*, il faut utiliser le *SDK Game*. C'est un kit de développement fourni par *Microsoft* qui fournit tous les outils et la documentation nécessaires à la programmation de *DirectX*. C'est pourquoi nous avons écrit pour vous un programme de base permettant d'utiliser *DirectX*, sans entrer dans les détails abscons. C'est une sorte de «squelette» que vous pourrez remplir avec vos propres routines.

Initialisation de DirectX

La programmation d'un logiciel avec *DirectDraw* se fait de la même manière qu'un programme *Windows* classique, c'est-à-dire qu'il faut gérer des tas de paramètres incompréhensibles pour réaliser la moindre action. Notre programme contient des routines qui s'occupent de cette initialisation pour vous. Quel que soit le mode graphique de votre système, les routines d'initialisation activent le mode graphique plein écran 640x480 en 256 couleurs.



▲ *Le logiciel de démonstration affiche plusieurs dizaines de milliers de fois par seconde le même sprite.*

Les routines de *DirectDraw* sont conçues pour travailler avec le système multitâche de *Windows 95*. Il est possible de fixer le degré de coopération entre *DirectDraw* et le reste du système. Nos routines activent le mode de coopération minimal, c'est-à-dire que *DirectDraw* réserve presque toute la puissance disponible pour ses propres besoins.

Les surfaces graphiques

Le système *DirectDraw* fonctionne essentiellement avec des surfaces graphiques. Une surface est une zone bitmap de taille quelconque qui peut se trouver sur l'écran ou dans la mémoire du PC. Lors de l'initialisation, nos routines fabriquent une surface de 640x480 pixels, associée à l'écran physique. Il est possible de dessiner des surfaces géométriques, de tracer des lignes, d'afficher du texte et de copier des images sur la surface.

Nous avons écrit différentes routines pour faciliter l'utilisation des surfaces. La première est *DDRAW_CreateSurface()* qui crée une surface de taille déterminée. Elle retourne une valeur de type *LPDIRECTDRAWSURFACE* qui est un pointeur sur la surface. Ce nom barbare nous est imposé par la bibliothèque de fonctions graphiques du *SDK Game*.

```
LPDIRECTDRAWSURFACE
DDRAW_CreateSurface (
      DWORD dwWidth,   // largeur
      DWORD dwHeight)  // hauteur
```

Vous trouverez la version 2.0 de DirectX sur le [illegible] prochainement

La surface peut avoir n'importe quelle taille. L'exemple suivant vous montre comment créer une surface de 640x480pixels :

```
LPDIRECTDRAWSURFACE pSurface;
int                 tx, ty;

tx = 640;
ty = 480;
pSurface = DDRAW_CreateSurface(
                    tx, ty);
```

La fonction *DDRAW_FillSurface()* permet de remplir n'importe quelle surface avec une couleur quelconque. Voici sa syntaxe :

```
void
DDRAW_FillSurface(
 LPDIRECTDRAWSURFACE pSurface,
 DWORD dwColor)
```

Voici un exemple qui remplit une surface avec une couleur quelconque :

```
LPDIRECTDRAWSURFACE pSurface;
DWORD               c;

c = 1;
DDRAW_FillSurface (pSurface, c);
```

La fonction *DDRAW_BmpToSurface()* crée une surface et charge une image *BMP* à l'intérieur. La taille de la surface est égale à celle de l'image. C'est une fonction importante car elle permet d'intégrer des éléments graphiques dans nos applications *DirectDraw*. Les paramètres de la fonction sont le nom de l'image et une variable tampon permettant de récupérer la palette de couleur de l'image. Cela peut servir au cas où il faut modifier la palette de couleurs de la surface principale pour la faire coller avec celle de l'image *BMP*.

```
LPDIRECTDRAWSURFACE
DDRAW_BmpToSurface (
 LPSTR              szFileName,
 LPDIRECTDRAWPALETTE *ppDDPal)
```

L'exemple suivant charge l'image «*sprite.bmp*» dans une surface créée pour l'occasion. Elle retourne un pointeur sur la nouvelle surface.

```
pSpriteSurface =
DDRAW_BmpToSurface(
      «sprite.bmp»,
      &pDDPalette);
```

La fonction *DDRAW_BltSprite()* permet d'afficher un sprite sur une surface. Le sprite est aussi une surface contenant une image bitmap. La routine d'affichage considère que la couleur 255 est «transparente», c'est-à-dire que tous les pixels de couleur 255 ne doivent pas être copiés sur la surface de destination. Le sprite est une image quelconque, dessinée sur un fond de couleur 255.

```
void DDRAW_BltSprite (
 LPDIRECTDRAWSURFACE pTrgSurface,
 LPDIRECTDRAWSURFACE pSpriteSurface,
 LPSIZE              psizeSprite,
 LPPOINT             pptSprite)
```

Le paramètre *pTrgSurface* est la surface de destination. Le paramètre pSpriteSurface est la surface contenant le sprite. Le paramètre *psizeSprite* est la taille du sprite. Le paramètre *pptSprite* contient la position d'affichage du sprite sur l'image.

Le programme

Notre programme de démonstration a été écrit pour évaluer les performances d'affichage de *DirectX*. Il charge un sprite en mémoire et l'affiche sur l'écran plusieurs millions de fois. Le but est de mesurer la vitesse d'affichage.

La routine principale du logiciel est *WinMain()*. Elle procède à différentes initialisations dont les plus

▲ *Voici l'image agrandie 6 fois du sprite. C'est un anneau dessiné sur un fond de couleur «transparente».*

importantes sont la création de la fenêtre de l'application et la déclaration de la routine *DXWndProc()* comme routine de traitement des messages de la fenêtre. La routine *WinMain()* contient également la boucle de messages du programme. Ce n'est pas une boucle classique. Normalement, une boucle de messages *Windows* attend qu'un message arrive pour le traiter. C'est différent avec notre programme dont la fonction est d'afficher des sprites sur l'écran. Chaque fois que la boucle de messages ne détecte aucun message, elle appelle la routine *DX1_Idle()* qui affiche un sprite sur l'écran. En d'autres termes, chaque fois que le système n'a aucune tâche urgente à traiter, il profite de son temps libre pour afficher un sprite.

La routine DXWndProc() est la routine de traitement des messages. Elle est utile pour récupérer les messages clavier et souris, mais aussi lors de la création et de la destruction de la fenêtre de l'application. Pour *Windows*, chaque application est nécessairement liée à une fenêtre. L'initialisation du programme doit se faire lors de la création de la fenêtre. Lorsque la fonction *WinMain()* utilise la fonction *CreateWindowEx()* ▶

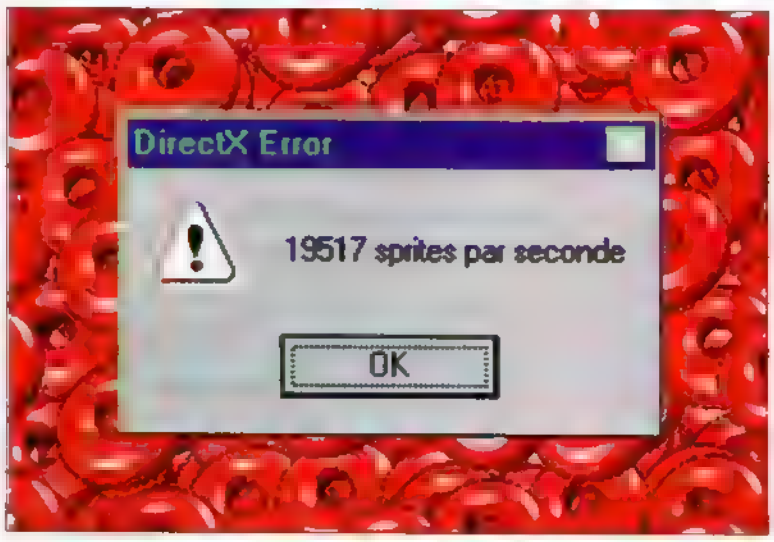
pour créer la fenêtre de l'application, le système émet le message *WM_CREAT* qui est intercepté et traité par *DXWndProc()*. Mais l'initialisation de *DirectDraw* ne peut se faire que lorsque la fenêtre est totalement initialisée ! C'est pourquoi le programme s'envoie un message *WM_USER* à lui-même pour se rappeler qu'il doit procéder à une initialisation. Le traitement de ce message lance l'exécution de la fonction *DX1_Init()*. Les autres messages traités par la fonction concernent la destruction de la fenêtre et les événements clavier et souris.

La fonction *DX1_Init()* initialise le programme. C'est une tâche double car elle doit s'occuper de l'initialisation normale du logiciel et d'activer le système *DirectDraw*. C'est la fonction *DDRAW_Start()* qui initialise *DirectXDraw*. Elle active le mode graphique plein écran 640x480 en 256 couleurs et indique à *Windows 95* que *DirectX* a une priorité d'exécution absolue sur le reste du système.

L'initialisation du programme n'est pas très compliquée. La routine remplit la surface écran *DirectDraw* avec la couleur 254 et charge ensuite l'image du sprite dans une petite surface *DirectDraw*, grâce à la fonction *DDRAW_BmpToSurface()*. Ce faisant, elle récupère la palette de couleurs du sprite et l'applique sur la surface écran. La gestion des palettes de couleurs est un point délicat dans la programmation des logiciels graphiques. En effet, que se passe-t-il si vous affichez un sprite possédant sa propre palette de couleurs sur une surface écran ayant une palette différente ? La réponse est une grosse bavure graphique ! En mode 256 couleurs, les images doivent avoir la même palette. C'est pourquoi le programme récupère la palette du sprite et l'utilise pour modifier la palette de l'écran.

Gestion du clavier

Comme nous l'avons vu plus haut, la fonction *DXWndProc()* reçoit tous les messages de l'application. Afin de nous simplifier la vie, nous avons dérivé les messages importants vers d'autres routines. Le programme exécute la fonction *DX1_OnKey()* chaque fois que Windows détecte un événement clavier et génère un message correspondant.



▲ *En appuyant sur la touche [F1], le programme indique la vitesse d'affichage des sprites.*

Les paramètres importants sont *fDown* qui indiquent si la touche a été enfoncée (*fDown=TRUE*) ou relâchée (*fDown=FALSE*) et *vk* qui contient le code de la touche. Il est possible de déterminer le code de la touche, grâce aux constantes *VK_* (*Virtual Keyboard*). La touche *VK_ESCAPE* correspond à la touche *[Echap]* et provoque la fin du programme, en envoyant le message *WM_CLOSE* (fermeture fenêtre). Ce message est intercepté par *Windows* qui efface en partie la fenêtre et lance le message *WM_DESTROY* pour indiquer *DXWndProc()* qui doit effacer les données du programme. Celui-ci utilise alors la fonction *DX1_Terminate()* pour désactiver *DirectX*, effacer le sprite et effacer la zone mémoire de la palette graphique.

La démonstration

Le programme active *DirectDraw* en mode plein écran 640x480, 256 couleurs, charge en mémoire un petit sprite de 48x46 et l'affiche sans cesse sur l'écran, à une position aléatoire. L'affichage du sprite est très impressionnant, on dirait une horde de créatures rouges grouillant sur l'écran. Cette impression est dûe à la grande vitesse d'affichage. Il suffit de presser sur la touche *[F1]* pour connaître le nombre de sprites affichés en une seconde. Avec un *Pentium 100*, on atteint les 19 500 sprites/seconde avec une carte graphique bas de gamme de type *Trio S3*. C'est plutôt impressionnant, étant donné qu'il s'agit d'un mode graphique 640x480 en 256 couleurs !

Utilisation du programme

Vous pouvez réutiliser notre programme pour créer vos propres applications *DirectX*. C'est une sorte de squelette multi-usage ! Toutes les fonctions commençant par *DX1_* peuvent êtres modifiées à volonté.
Le source doit être compilé avec un compilateur 32 bits, comme le *Visual C 4.0* ou le *Borland C 5.0*. Vous n'avez pas besoin du *SDK Game* pour compiler ce programme. Nous avons extrait de ce kit le fichier *ddraw.h* contenant la déclaration de toutes les fonctions de *DirectDraw*.

Christophe Novak

Un de ces numéros vous manque ?

N° 1

Astuces - Routines - Programmation - Formation - Les meilleurs Sharewares - Jouer à plusieurs en réseau ou par modem.

Disquette

Un kit de compression - L'anti-virus FProt 2.17 - L'ensemble des routines du magazine + documentation en français du compacteur LHA.

N° 2

Formation - Sharewares - Visual Basic - Programmation - C++ - Astuces - Jeux - Internet facile et pas cher en quelques ruses.

Disquette

Zip et Unzip avec les sources - NSV : un «viewer» parmi les plus rapides du monde - Les routines, programmes et listings du magazine.

N° 3

Épuisé

N° 4

Visual Basic : introduction à OLE, fonction de dessin, espionner Windows - C++ - Delphi : gérer les polices, le modèle document/vue, les fonctions virtuelles, initiation à la POO.

Disquette

Utilitaires Dos, librairies QBasic, générateur de fractales et sources C, routines et sources TP Windows...

N° 5

Delphi : construire une application professionnelle.

C++ - Pascal : librairies d'objets graphiques, programme de dessin vectoriel.

Visual Basic : éditer la «registry», créer des outils de dessin, optimiser les tris.

Disquette

Librairies graphiques Windows et sources C, générateur d'organigrammes et symboles, programme de dessin QBasic...

N° 6

Visual Basic : cacher des messages secrets dans vos programmes, programmer votre propre packman.

Langage C : un ovni pour Windows 95, créer votre casse-brique.

QBasic : des sprites en ombres chinoises.

Disquette

Les routines, programmes et listing du magazine, un driver VESA pour les modes graphiques haute résolution.

N° 7

Deux concours pour les créateurs de jeux vidéos.

Langage C : gérer le mode VGA 320x200, 256 couleurs en Borland C, création d'un jeu C.

Visual Basic : les secrets de l'instruction, PainPicture, création d'un jeu.

Débutant : les conseils pour apprendre à programmer.

Disquette

Les routines, programmes et listings du magazine, le Liberty Basic, un langage simple pour écrire des programmes Windows.

Pour commander les numéros: adressez un chèque de 49 francs + 11 francs de participation aux frais de port par exemplaire à

J.D. PRESS, 24 bis, rue du Gabon
75012 Paris

(indiquez au dos du chèque le ou les numéros choisis, si l'adresse indiquée au recto du chèque ne correspond pas à l'adresse de livraison, précisez-la sur le verso du chèque).

Soyez rusé...

Abonnez-vous...

20% d'économie

OFFRE SPECIALE D'ABONNEMENT

468 F*
au lieu de 588 F

OUI, je m'abonne à PC Futé
pour 12 numéros au prix exceptionnel de 468 F.

Je joins un chèque de 468 F à l'ordre de J.D. Press.

Nom : ..... **Prénom :** .....

Adresse : .....

.....

Code postal : ..... **Ville :** .....

Date : ..... **Téléphone :** .....

A remplir et à retourner accompagné de votre règlement (par chèque exclusivement) à :
J.D. Press, 24 bis, rue du Gabon, 75012 Paris
Vous pouvez recopier sur papier libre ou photocopier ce bon.

**offre valable pour la France métropolitaine uniquement. Pour les DOM/TOM, la CEE et les autres pays nous consulter.*

simple est d'utiliser un tableau dont chaque case correspond à une position de l'écran. La valeur 0 correspond à l'absence de cube (au cube vide en fait !). C'est le tableau *jeu[]* qui remplit ce rôle dans le programme :

```
const
 MaxLignes = 10;
 MaxColonnes = 8;

var
 jeu : array[1..MaxLignes,
             1..MaxColonnes]
             of integer;
```

Pour savoir si un cube se trouve bloqué à une position de l'écran, il suffit de regarder la valeur correspondante du tableau *jeu[]*.

```
ligne := 2;
colonne := 5;
test := jeu[ligne, colonne];
```

Pour savoir si un cube bloque la chute du cube courant, la fonction *test_bas* calcule la position de chute et regarde si quelque chose se trouve à cet emplacement. Si la position est libre, elle retourne la valeur true. Elle retourne la valeur false dans le cas contraire.

```
ligne:=ligne_cube+1;
colonne:=colonne_cube;
if jeu[ligne,colonne]<>0 then
begin
  test_bas := false;
  exit;
end;
test_bas:=true;
```

La routine *fin_chute* est appelée lorsque le cube courant est bloqué. Elle mémorise la position du cube dans le tableau *jeu[]* et appelle la routine *nouveau_cube* pour afficher un nouveau cube en haut de l'écran.

```
procedure fin_chute;
begin
 (* MEMORISATION CUBE *)
 jeu[ligne_cube, colonne_cube]
    :=cube_courant;
 (* GENERATION NOUVEAU CUBE *)
 nouveau_cube;
end;
```

Gestion du clavier

Pendant la chute du cube, le joueur peut influencer sa trajectoire avec les flèches du clavier. Pour cela, il faut que le programme soit capable de détecter les événements claviers. Théoriquement, les événements claviers sont interceptés par les composants concernés (*boite Edit* par exemple). Heureusement, il est possible de forcer une fiche à intercepter les événements claviers en activant sa propriété *KeyPreview*.

```
Form1.KeyPreview := true;
```

La procédure *Tform1.OnKeyDown* est activée chaque fois que la fiche détecte une pression sur une touche. Le paramètre utile pour nous est *Key* qui contient le code de la touche pressée. Il est possible d'identifier la touche concernée grâce aux constantes *VK* (*Virtual Keyboard*). Il existe une constante *VK* pour chaque touche du clavier. Par exemple, la constante *VK_LEFT* correspond à la flèche gauche et *VK_RIGHT* à la flèche droite. Notre procédure utilise ces constantes pour détecter les flèches du clavier et la touche *[Echap]*.

```
procedure TForm1.OnKeyDown(
            Sender: TObject;
              var Key: Word;
        Shift: TShiftState);
begin
 if Key=VK_LEFT
      then aller_gauche;
 if key=VK_RIGHT
      then aller_droite;
 if key=VK_DOWN
      then chute_cube;
 if key=VK_ESCAPE
      then halt;
end;
```

La routine *aller_gauche* déplace le cube d'une colonne vers la gauche. La routine *aller_droite* fait la même chose vers la droite et la routine *chute_cube* fait tomber le cube, jusqu'à ce qu'il touche un autre cube ou la fin de l'écran. Le code de ces routines est similaire à celui de *aller_bas*, à part le sens de déplacement. Lorsque le programme détecte l'appui sur la touche *[Echap]*(constante *VK_ESCAPE*), il utilise l'instruction *halt* pour stopper l'exécution du programme.

Pour finir

La disquette du mois contient une première version du jeu. Les cubes tombent et les flèches du clavier modifient la position du cube courant. En revanche, le logiciel ne gère pas la détection et l'effacement des alignements de cubes identiques. C'est un sujet complexe qui nécessite un article complet. Nous verrons une version totalement opérationnelle du programme dans le prochain numéro de PC FUTE.

Marc Lenoir

Remarque

Ce programme a été écrit de manière classique, sans utiliser la programmation objet. Si un lecteur a le courage de réécrire le logiciel en mode objet, nous serons heureux de mettre la nouvelle version sur la disquette.

S O N D A G E

participez à l'amélioration de votre magazine

Vous pouvez cocher plusieurs réponses pour une même question.

Qui êtes-vous ?

❑ Programmeur professionnel
❑ Programmeur amateur
❑ Autre (*precisez*) .....

Votre niveau de connaissances

❑ Néophyte complet
❑ Débutant
❑ Expérimenté
❑ Très expérimenté
❑ Autre (*precisez*) .....

Centres d'intérêt

❑ Initiation
❑ Graphisme
❑ Jeux
❑ Programmation système
❑ Utilitaire
❑ Bureautique
❑ DOS
❑ OS/2
❑ Windows 3.1
❑ Windows 95
❑ Autre (*precisez*) .....
.....

Langages

Veuillez préciser les versions des langages utilisés

❑ Qbasic version .....
❑ Quick Basic version .....
❑ Autre Basic version .....
❑ Visual basic version .....
❑ Turbo Pascal version .....
❑ Autre Pascal version .....
❑ Delphi version .....
❑ Visual C* version .....
❑ Borland C* version .....
❑ Autre C* version .....
❑ Assembleur version .....
❑ Autre langage nom .....
version .....

*Programmez-vous en C ❑ ou en C++ ❑ ?

Que préférez-vous ?

❑ Des exemples de programmes
❑ Des articles théoriques
❑ Des articles d'initiation
❑ Autre (*precisez*) .....

Votre configuration

Micro-ordinateur .....
Microprocesseur .....
Mémoire .....
Disque dur .....
CD-ROM .....
Carte graphique .....
Carte son .....
Autres équipements .....
Système d'exploitation .....
Logiciels .....
.....
.....
.....

Vos coordonnées

Nom .....
Prénom .....
Adresse .....
Code postal .....
Ville .....
N° de tél ..... Age .....

Votre activité

❑ étudiant ❑ enseignant
❑ employé ❑ cadre
❑ informaticien ❑ directeur/resp. de service
❑ Autre (*precisez*) .....

Votre employeur

Effectifs ❑10 X ❑10-50 ❑ 50-200 ❑ >200 X
Votre service .....

Acceptez-vous de voir vos coordonnées publiées dans notre annuaire des programmeurs ?
OUI ❑ NON ❑

Avez-vous des suggestions ou des thèmes d'articles à proposer ?
.....
.....
.....
.....
.....
.....
.....

Vous manquez de place ? Continuez sur papier libre et agrafez-le au présent questionnaire.

Découpez ou photocopiez la page remplie et retournez-la à l'adresse suivante :
J.D. Press - 24 bis, rue du Gabon - 75012 Paris

BIEN SÛR QU'IL Y A DELPHI, MAIS IL N'Y A PAS QUE DELPHI ! DONC, 3 OUVRAGES POUR DELPHI ET 3 AUTRES POUR VOUS PERMETTRE DE PROGRAMMER PLUS FACILEMENT.

Des livres pour programmer 6 livres pour évoluer

Le Zen de l'optimisation du code

Les programmeurs utilisant le C ou le C++ savent déjà que les programmes utilisant ces langages peuvent être réduits au maximum. Mais réduction de code source ne signifie pas optimisation du code. Ce livre est une compilation de techniques pour mieux programmer et surtout programmer plus vite. Ces techniques ont déjà été publiées dans les magazines PC Techniques et Dr. Dobb's journal. En parcourant les 22 chapitres (plus de 600 pages), vous évoluerez dans le monde de l'interaction C et assembleur. L'auteur insiste bien sur l'importance de l'assembleur pour mieux optimiser votre code. Il va même jusqu'à expliquer le Pentium et ses règles. C'est pour cela qu'il est préférable d'en avoir un pour compiler et exécuter les exemples du livre qui sont d'ailleurs sur le CD-ROM qui accompagne cet ouvrage. Sur celui-ci, vous trouverez aussi le code source du «Timer Zen» qui est un programme pour mesurer les performances de votre futur code. Cet ouvrage s'adresse à des personnes connaissant déjà la programmation et surtout, possédant un déboggueur tel que CodeView et bien sûr un compilateur C ou C++. Il faut aussi que ces personnes soient «zen» pour appréhender les exemples fournis. Même si le code est soigneusement écrit, il est néanmoins fait pour des initiés.

Auteur : Michael Abrash.
Editeur : SYBEX.
Prix : 98 F avec CD ROM.

Le Meilleur : Programmation Jeux 3D

1 000 pages pour appréhender la programmation des jeux. Ce livre vous dévoile maints secrets sur les techniques avancées utilisées par des jeux comme DOOM. Outre les sujets types, tels que graphiques, sons, environnement Windows 95, cet ouvrage traite aussi de la protection de vos programmes : mode anti-débogage, mots de passe...
Bien sûr, de telles techniques passent par la connaissance du monde des "crakers" que vous survolerez dans les chapitres 10 et 11. Ce livre peut être lu dans n'importe quel ordre, suivant l'information que vous recherchez (comme le format des images GIF ou des fichiers MOD), mais il ne faut absolument pas passer le premier chapitre qui vous donne un glossaire des termes les plus importants (WHQ, EHQ, Messy...) à savoir. Puis vous aurez un cours sur l'assembleur. Il faut donc avoir quelques connaissances de programmation. Bizarrement, le langage choisi pour mieux appréhender les exemples du livre n'est pas le langage C, mais le PASCAL. D'autres chapitres vous aideront à accélérer la vitesse de vos jeux, en gérant mieux la mémoire par exemple, ou en optimisant certaines routines par des algorithmes mieux conçus. Les cartes sonores sont bien traitées et les cartes Sound Blaster et Gravis sont détaillées. Le CD-ROM vous donne des outils indispensables pour vous aider dans votre développement. Il contient la totalité des exemples du livre. Cet ouvrage est destiné aux programmeurs ayant déjà une expérience dans la programmation des jeux et qui veulent la parfaire.

Auteurs : Equipe Data Becker.
Editeur : Micro Application.
Prix : 149 F avec CD-ROM.

Grand livre HTML

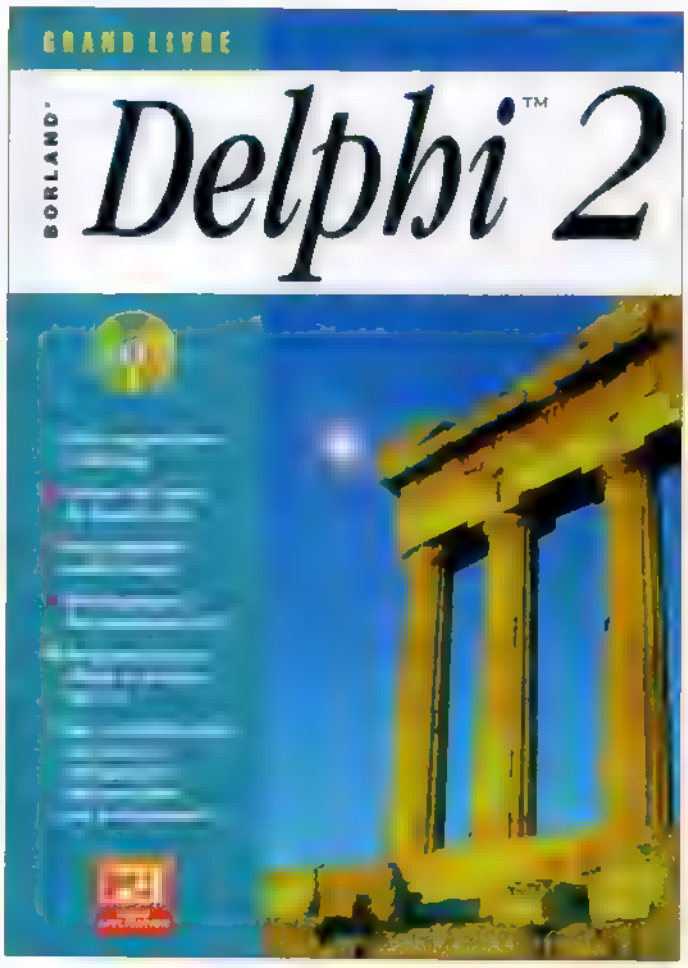
Les langages HTML et SGML deviennent de plus en plus incontournables, grâce à l'essor que prend Internet. Ce livre vous présente donc toutes les caractéristiques des éditeurs graphiques et browsers sous Windows 95. Sur le CD-ROM se trouve la majeure partie des éditeurs les plus connus (Hotmetal, Hotdog, homepage creator, Microsoft Internet Assistant ...). Parmi les sujets traités, vous trouverez l'historique du SGML, les structures des documents sous HTML 2.0 et 3.0, les différentes extensions HTML (en ce moment les frames font fureur), l'adressage sous Internet (adresse IP ...), comment configurer un serveur HTTP pour Windows, ainsi qu'un serveur FTP. L'avantage certain de ce livre est la façon claire et très illustrée d'appréhender les sujets. Les snaps d'écran sont nombreux ce qui permet de mieux suivre le déroulement de l'installation d'un browser ou d'un serveur. Un autre point important : le glossaire. Il est bien détaillé et comporte tous les termes qui sont nébuleux lors de la lecture de certains chapitres. Les 500 pages et les 10 chapitres s'adressent donc à tous, avec cependant une petite restriction : pour mieux profiter de ces nombreuses connaissances, il est préférable d'avoir un modem et Windows 95.

Auteur : Joachim SCHWARTE.
Editeur : Micro Application.
Prix : 195 F avec CD ROM.

Grand Livre Delphi2

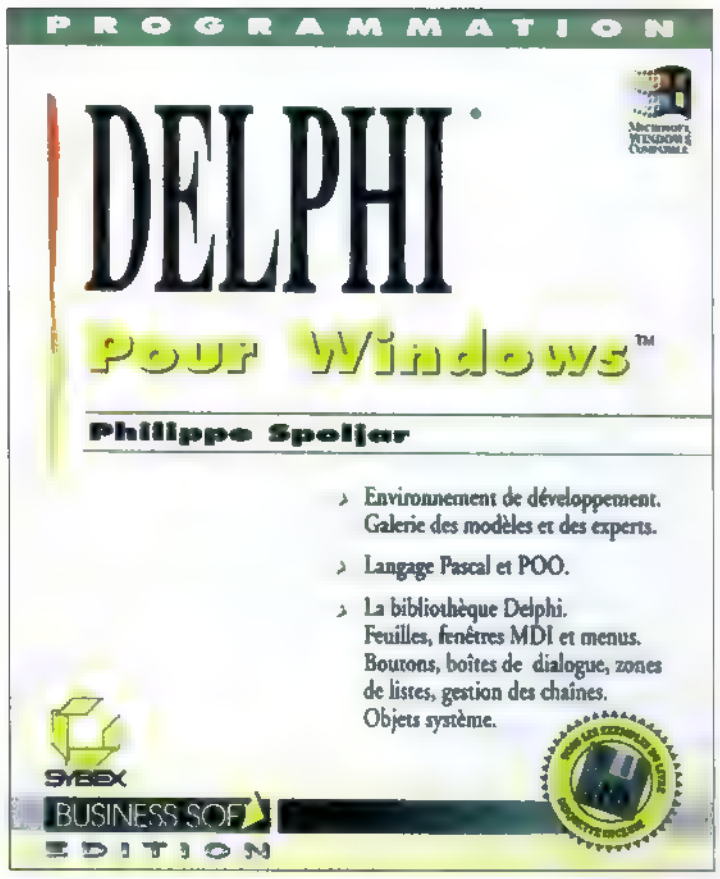
La programmation Windows devient de plus en plus accessible avec Delphi2. Ce livre vous en convaincra sûrement, puisqu'il vous démontrera d'une façon rapide et efficace la façon la plus simple de programmer. Pour cela, de nombreux exemples sont fournis et pour vous éviter de les taper, le CD-ROM les reprend. Sur ce CD-ROM, vous trouverez aussi la conférence des développeurs de Borland Delphi (1996), la base de connaissance de Borland, des librairies et de nombreux autres exemples. Parmi les sujets abordés, vous ferez vos premiers pas sous Delphi et vous terminerez par le savoir-faire Delphi. Entre temps, vous aurez été formé par un rapide cours sur Delphi. Ce cours est destiné aux débutants. La programmation Objet est expliquée, ainsi que le développement de bases de données. Pour ceux qui ont des connaissances réseau, l'aspect client serveur est abordé. Enfin, le développement de type professionnel (hiérarchie de classes, multi-thread...) termine le livre. Le débutant et l'initié y trouveront donc une multitude d'informations. Cependant, il faut quand même posséder le logiciel Deplhi pour lui adjoindre ce livre comme compagnon.

Auteur : Burda Färber.
Editeur : Micro Application.
Prix : 195 F avec CD ROM.



Delphi pour Windows

Bien qu'il y ait plusieurs milliers de lignes de programmes en Delphi, ce livre n'est pas destiné aux personnes ayant déjà une expérience en Delphi mais plutôt aux débutants. qui pourront ainsi maîtriser très facilement l'environnement et le développement sous Delphi. Tous les menus sont présentés et le langage Pascal est décortiqué. De la simple structure de base jusqu'à l'introduction de la programmation Objet, tout le langage Pascal est abordé. Après ces nombreux chapitres, vous accéderez à la bibliothèque d'objets de Delphi.



Comme ce livre est destiné à des personnes ayant peu de connaissances en programmation, seule une disquette est fournie, ce qui est amplement suffisant pour commencer. Pour ceux qui voudraient en savoir plus, ils trouveront une étude détaillée du coeur de la bibliothèque de Delphi (la VCL Visual Component Library, entre-autres). Cet ouvrage s'adresse à ceux qui viennent d'acheter Delphi et qui ne connaissent que très peu le langage PASCAL.

Auteur : Philippe Spolja.
Editeur : SYBEX.
Prix : 248 F avec disquette.

Delphi2 Secrets d'experts

Ce livre fait concurrence au «Grand Livre DELPHI2». Bien qu'il soit plus cher, ses 1200 pages justifient la différence de prix. En effet, un chapitre supplémentaire est consacré à l'introduction à la programmation CGI (Internet) sous Delphi. Un CD-ROM est livré dans sa version originale américaine. Tous les exemples sont compilés en 16 ou 32 bits. Cet ouvrage est aussi bourré de trucs et astuces dont vous avez besoin pour créer des applications professionnelles. Parmi les différents sujets abordés, voici une liste des plus importants : EDI, pointeurs, composants visuels ou non, travailler avec Interbase, automatisation OLE et COM, les SDK et les capacités multimédias avancées. D'autres chapitres vous expliqueront les notions d'API et de DLL sous Windows, mais aussi les liens avec SQL et ReportSmith. Bien sûr, l'aspect client/serveur est aussi abordé. L'auteur, grand spécialiste de Delphi, nous indique sur le CD-ROM les outils qu'il utilise : Kit de développement HTI Map de Horizon Technology, Light Lib Image de DFL Software et pleins d'autres petits outils nécessaires pour une programmation optimale. Ce livre est donc une bible pour tous ceux qui veulent vraiment développer en Delphi.

Auteur : Charles Calvert.
Editeur : S&SM.
Prix : 329 F avec CD-ROM.

Roch Lombardon

VOUS VOULEZ FAIRE CONNAÎTRE VOS CRÉATIONS ? VOUS AVEZ QUELQUE CHOSE À DIRE AUX AUTRES LECTEURS ? VOUS AVEZ ÉCRIT LE PROGRAMME DU SIÈCLE ? CETTE RUBRIQUE EST FAITE POUR VOUS! ENVOYEZ-NOUS VOS CRÉATIONS, NOUS LES PUBLIERONS SUR NOTRE DISQUETTE.

Vos programmes

Ce mois-ci, la disquette contient 3 programmes écrits par des lecteurs. Nous en avons reçu davantage, mais ils étaient trop volumineux pour tenir sur la disquette. Pour des raisons purement techniques dues à la taille limitée de la disquette, nous ne pouvons pas consacrer une place trop importante à cette rubrique.
Envoyez-nous des fichiers textes et des listings plutôt que des programmes exécutables faisant 500 ou 600 Ko !

Dans l'idéal, nous aimerions recevoir des fichiers ne faisant pas plus de 100 Ko, une fois compilés ! Ce n'est pas gênant de mettre deux images BMP de 300 Ko, un listing de 75 Ko et un texte de 20 Ko si le tout n'occupe pas plus de 100 Ko sur notre disquette! Les langages utilisés peuvent être le *Visual Basic*, *Delphi*, *Qbasic*, *Turbo Pascal*, le *Langage C* ou un langage de programmation comme le *Liberty Basic* fourni avec PC FUTE n°7. Les programmes sources doivent être fournis, surtout si vous utilisez un langage comme *Delphi*, qui génère des programmes de plusieurs centaines de Ko ! Si possible, écrivez un petit texte expliquant comment fonctionne le programme et donnez votre nom et votre adresse à l'intérieur.

Le programme Infernal

Il s'agit d'un programme mystérieux écrit par *Thomas GEMPP*. Le logiciel n'est pas fourni sous la forme d'un source, mais d'un texte stocké dans un fichier au format *wri*. Il faut utiliser les fonctions *Couper/Coller* d'un traitement de textes comme *Write* ou *WinWord* pour récupérer le source dans un fichier texte. Le programme existe dans une version *Turbo Pascal* et une version *C*. A la lecture, ce programme semble vraiment très curieux, d'autant plus que l'auteur insiste sur le fait qu'il n'y a aucune valeur numérique dans le programme. Arriverez-vous à percer le secret ?

Le programme Passe

La sécurité informatique préoccupe beaucoup de monde. C'est pourquoi un lecteur nous a envoyé un petit programme écrit en *VB 4.0* permettant de sécuriser l'accès à Windows avec un mot de passe. Le produit est très intéressant, car l'auteur a également écrit un petit article expliquant comment fonctionne son logiciel. Le source est en *VB 4.0* mais la conversion en *VB 3.0* ne doit pas poser de problèmes.

Le module Lutin

Il s'agit d'une petite bibliothèque de fonctions permettant d'afficher des sprites en *Delphi*. Elle est fournie avec un programme d'exemple affichant un camion, un hélicoptère et un parachutiste en superposition sur un décor. Nous devons cette création à *Jean-Marie SCAYA*, qui a aussi eu la bonne idée de taper un petit texte explicatif sur l'affichage des sprites.

Quelques regrets

Nous regrettons de ne pas pouvoir mettre sur la disquette les programmes de *Stéphane Trichet*. Il s'agit d'un utilitaire de désinstallation des logiciels *Windows* et d'une horloge avec alarme. Ces deux programmes *Delphi* sont trop gros pour être mis sur la disquette et l'auteur nous a seulement envoyé les programmes exécutables sans les sources. C'est dommage! La prochaine fois, Stéphane envoie-nous les sources si tu veux voir tes créations sur la disquette !

Nous regrettons également la non-parution du programme *Relief* de *Jacques François Wadel*. C'est un programme de démonstration graphique écrit en *VB 3.0*, inspiré de l'article publié dans PC FUTE n°6 sur l'instruction *MouseMove*. Il s'agit de l'image de deux grenouilles dont le regard suit les déplacements du curseur souris. Le programme est simple, mais les données graphiques prennent plus de 600 Ko, même compressées ! Nous mettrons ce pro-gramme sur le CD-ROM Hors Série de PC FUTE qui sort prochaînement.

La rédaction

Cette technique rend le fantôme beaucoup plus agressif. La seule manière de le contrer est de manoeuvrer rapidement pour quitter son champ de vision, et d'aller dans un autre endroit de l'écran. Mais cela n'est efficace que contre un seul fantôme! Avec deux ou trois adversaires, il faut réagir très vite pour éviter la catastrophe!

Gestion du fantôme

La vitesse de déplacement du fantôme doit être la même sur tous les PC. La manière la plus simple de réaliser cela est d'utiliser un contrôle *Timer*, dont chaque activation provoque l'exécution de la routine de déplacement du fantôme. Cela permet de synchroniser la vitesse sur tous les PC et de permettre un réglage facile de la vitesse du fantôme, en agissant sur la fréquence d'activation du *Timer* (propriété *Interval*). Le programme utilise 3 *Timers* pour ces besoins propres. Celui associé à la gestion du fantôme porte le nom de *Minuterie2*. Sa période d'activation dans le programme est déterminée par la constante *VITESSE_FANTOME*. Elle contient la valeur 300 qui correspond à un appel de la fonction *gestion_fantome()* tous les 300 ms. Voici l'algorithme de la routine :

```
LE PACMAN EST-IL A COTE ?
SI OUI=> DESTRUCTION PACMAN
         QUITTER ROUTINE

LE PACMAN EST-IL EN VUE ?
OUI => CHANGER DIRECTION

DEBUT BOUCLE
 CALCUL POSITION FANTOME
 EST-CE UNE POSITION POSSIBLE ?
 OUI => SORTIE BOUCLE
 NON =>CALCUL NOUVELLE DIRECTION
       RECOMMENCER BOUCLE
FIN BOUCLE

DEPLACEMENT FANTOME
```

Activation du fantôme

Au début de la partie, le fantôme reste immobile dans son donjon pendant plusieurs secondes. Cela correspond la position où le créateur du tableau a placé l'image du fantôme. Le contrôle *Timer Minuterie3* fixe le temps d'attente de la créature, définie par la constante *ATTENTE_FANTOME*. Quand le *Timer Minuterie3* s'active, il entraîne l'exécution de la fonction *apparition_fantome()* qui fait apparaître le monstre à un emplacement aléatoire de l'écran. Il écrit ensuite la valeur *FANTOME_ATTAQUE* dans la variable et *at_fantome*. C'est une variable globale qui contient l'état courant du spectre (*FANTOME_PASSIF*=pas d'action; *FANTOME_ATTAQUE*= recherche et destruction du pacman; *FANTOME_FUITE* = fuite devant un pacman rouge capable de manger les revenants).

Quelques améliorations possibles

Pour avoir plusieurs fantômes il faut utiliser un tableau pour stocker les informations définissant chaque fantôme (position et direction de déplacement). Ce n'est pas la peine d'utiliser un contrôle *Timer* pour chaque fantôme. Il suffit d'un *Timer* unique pour appeler une routine gérant les mouvements de tous les revenants. Vous pouvez personnaliser les fantômes en jouant sur le calcul des nouvelles trajectoires. Certains fantômes peuvent privilégier les lignes et d'autres les colonnes.

Dans le véritable jeu de pacman, le pacman peut manger des gommes rouges qui le métamorphosent et lui donnent le pouvoir de manger les fantômes. Ceux-ci retournent alors à leurs positions de départ et attendent un moment avant de repartir. Vous pouvez gérer cela avec un *Timer* supplémentaire qui est initialisé au moment où la gomme rouge est mangée, et qui compte le temps pendant lequel le pacman possède le pouvoir. Pendant ce temps-là, les fantômes doivent fuir le pacman. C'est réalisable en modifiant la vision artificielle des fantômes de manière à ce qu'ils prennent la direction opposée au pacman dés qu'ils le voient.

Il est possible de programmer un système de bonus. Il suffit pour cela d'utiliser un contrôle *Timer* s'activant toutes les 10 ou 15 secondes et faisant apparaître un bonus à un endroit quelconque du labyrinthe. Vous pouvez utiliser la même technique que la fonction *apparition_fantome()* pour trouver une case non occupée par un mur, un fantôme ou le pacman. Un autre *Timer*, ou le même peut s'activer au bout de quelques secondes et faire disparaître le bonus, si le joueur ne l'a pas pris.

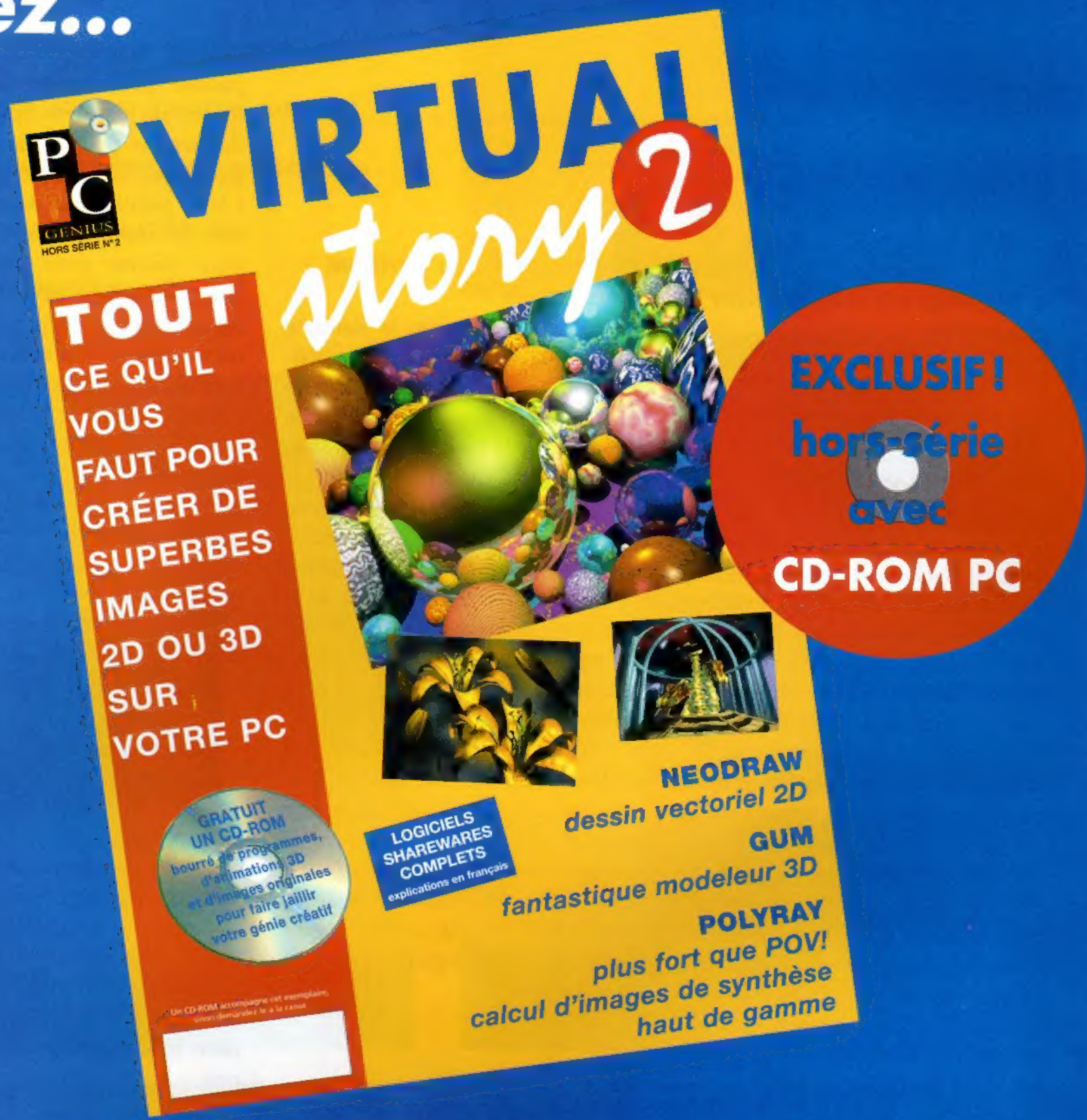
Pour finir

Voilà, c'est fini ! Le jeu n'est pas complet mais vous connaissez maintenant toutes les techniques nécessaires à la création d'un jeu complet. C'est à vous de jouer et de créer votre propre pacman ! Envoyez-nous vos créations !

John Lynch

Faites jaillir votre créativité

lisez...



Le 1er guide interactif de la création d'images de synthèse

Bon à découper ou à photocopier

❑ **oui**, je commande le **hors-série PC Génius n°2 avec CD-ROM** au prix de 75F + 11F de participation aux frais de port. Je joins un chèque de 86F à l'ordre de **J.D. Press** et j'envoie le tout à **J.D. Press, 24 bis, rue du Gabon, 75012 Paris**.

Nom ..... Prénom .....

Adresse .....

Code Postal ..... Ville .....

Pour installer le contenu de la disquette du mois, il faut l'insérer dans le lecteur de disque et lancer l'exécution du programme *INSTALL*.

• Sous DOS il faut taper le texte *INSTALL* et presser sur la touche *[Entrée]*.

• Sous Windows, il suffit de double-cliquer sur le fichier *INSTALL.BAT*.

La disquette s'installe alors automatiquement sur le lecteur C:. Il faut au moins 4,4 Mo de libre pour que l'installation se fasse correctement. Les fichiers s'installent dans le répertoire ***C:\PCFUTE\PCFUTE08***.

Visual Basic

Le répertoire ***\VB*** contient les listings des articles d'initiation sur le *VB*, et des exemples de programmes et de fichiers *VBX*. Le détail de ces fichiers est présenté à la page 34 du magazine. Le répertoire ***\Pacman*** contient les versions *VB 3.0* et *VB 4.0* de notre pacman, ainsi qu'un éditeur permettant de créer de nouveaux tableaux.

Delphi

Le répertoire ***\Delphi*** contient 4 programmes écris en *Delphi*. *Bmpscrol* est un petit composant permettant de faire scroller une image *Bmp* dans une fenêtre. *Dialer20* est un composant pour composer des numéros de téléphone directement dans un programme. *Supreview* est un playeur multimédia permettant d'écouter des sons (*Waves, Midi*), de charger des images *Bmp*, des fichiers textes et de jouer des animations *Avi* et *Flc*. Si vous voulez intégrer des animations dans vos programmes, il faut étudier ce listing à fond. Et *Imagine* est une DLL contenant des fonctions permettant de charger des images *Bmp, Gif, Pcx, wmf, jpg, mac*, et *tga* dans vos programmes *Delphi*. Le répertoire ***\Columns*** contient le source *Delphi* de notre article sur la programmation d'un jeu en *Delphi*.

Qbasic

Le répertoire ***\Qbasic*** contient les listings de notre article sur les trucs et astuces en *Qbasic*, ainsi que les listings de 2 petits jeux. Ces logiciels sont d'un intérêt moyen, mais leurs listings sont intéressants à étudier.

Garantie anti-virus

Cette disquette est garantie contre tous les virus connus.

ECHANGE DE DISQUETTE - GARANTIE ILLIMITÉE

Si la disquette de ce magazine présente un problème de duplication, nous l'échangeons GRATUITEMENT; Toute mauvaise utilisation (écriture sur la disquette...) annule cette garantie !

Langage C

Le répertoire ***\Civ*** contient les listings et les images de l'article sur la programmation d'un jeu de simulation stratégique de type *Civilisation*. Le répertoire ***\Graphvga*** contient les listings de l'article sur la structure de la mémoire écran *VGA*. Et le répertoire ***C\Loadbmp*** contient les listings de l'article sur le chargement d'une image *BMP* en mode graphique *VGA* 320x200. Tous ces listings existent en version *Visual C* et *Borland C*.

DirectX

Le répertoire ***\Sdkgame*** contient le listing *C* d'un petit programme de démonstration du système *DirectX* pour *Windows 95*. Cette démonstration ne peut fonctionner que si vous avez *Windows 95* et *DirectX* déjà installés sur votre machine.

Lecteurs

Le répertoire ***\Lecteurs*** contient les créations de nos lecteurs. Il n'y a que 3 produits car la plupart des créations que nous avons reçues étaient trop volumineuses pour la disquette. Il y a un système de mot de passe en *Visual Basic*, une bibliothèque en *Delphi* pour gérer des sprites et un programme mystérieux en *Langage C*.

Une remarque technique

Par manque de place, nous n'avons pas mis sur la disquette les programmes exécutables de grandes tailles, mais seulement leurs listings. C'est surtout valable pour *Delphi* dont les sources sont très compacts, et les programmes *EXE* assez volumineux.

La rédaction

Allez plus loin avec votre PC

Attention : ceci n'est pas un guide, mais un outil... Vous êtes débutant en programmation ou programmeur expérimenté. Vous utilisez Windows ou vous êtes resté fidèle au DOS. Dans tous les cas, vous avez besoin d'informations. Nous avons des articles d'initiation et des articles plus complets, conçus pour intéresser tous le monde.

Visual Basic

- Améliorez les performances graphiques de VB 3.0 et VB 4.0 en utilisant les DLL de Windows et réalisez des effets spéciaux graphiques sur mesure.
- Découvrez comment jouer des fichiers sonores Wav avec le VB 3.0 ou le VB 4.0 !
- Utilisez les contrôles Timers du VB pour gérer les actions des monstres dans un jeu d'action.

Delphi

- Quel est le meilleur langage de programmation pour Windows ? Delphi ou le Visual Basic ? Nous avons soumis Delphi à un redoutable test et le résultat est surprenant !
- Utilisez les fonctions graphiques de Delphi pour créer des jeux graphiques.
- Chargez des images GIF, PCX, JPG et TGA dans vos programmes Delphi.

Langage C

- Accélérez les performances de vos programmes en manipulant directement la mémoire écran VGA.
- Chargez des images BMP 256 couleurs dans vos programmes.
- Créez un jeu de simulation stratégique de type Civilisation.

Et aussi

- Programmez le système graphique DirectX du SDK Game pour réaliser des affichages graphiques ultra-rapides sous Windows 95.
- Découvrez comment fonctionnent les jeux d'aventure.
- Lisez nos trucs et astuces sur le Qbasic.

Concours

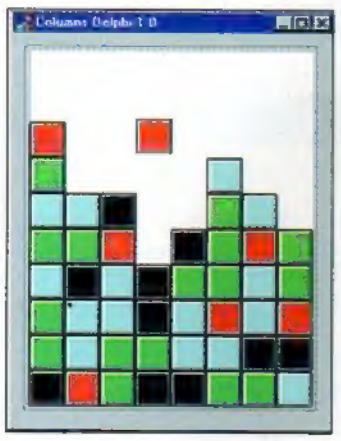
- Participez à notre grand concours sur la création du meilleur jeu vidéo.

La disquette PC FUTE

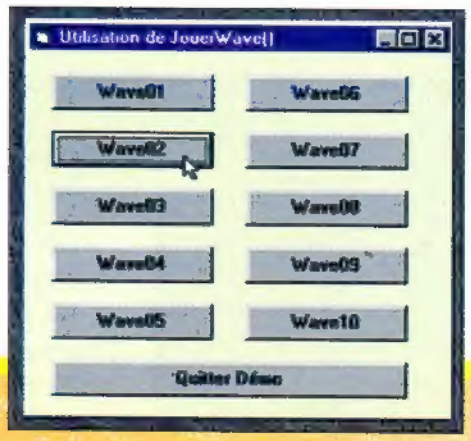
Plutôt que de taper les listings à la main, il est plus simple de récupérer les listings sur la disquette. Elle contient également de nombreux fichiers utiles (des sources et des composants pour le VB et Delphi, des programmes en Qbasic, des fichiers sonores Wav, etc.). Vous pouvez aussi y trouver les créations de nos lecteurs, et peut-être bientôt les vôtres ?

Utilisez le langage C pour créer un jeu de simulation stratégique de type «Civilisation».

Découvrez la puissance des fonctions graphiques de Delphi



Découvrez comment jouer des fichiers sonores Wave en Visual Basic 3.0 et 4.0.



Utilisez les fonctions graphiques DLL avec le Visual Basic.